Studies on the Application of Formal Semantics to English-Japanese Machine Translation

Toyoaki NISHIDA

Doctoral Thesis

Department of Information Science
Faculty of Engineering
Kyoto University

October 1983

Studies on the Application of Formal Semantics to English-Japanese Machine Translation

Toyoaki NISHIDA

Doctoral Thesis

Department of Information Science
Faculty of Engineering
Kyoto University

October 1983

Studies on the Application of Formal Semantics to English-Japanese Machine Translation

Toyoaki NISHIDA

Abstract

In this thesis, we attempt to figure out a flexible and sound framework for machine translation. Generally, language translation is a highly sophisticated task which requires wide range of knowledge about language and real world. To make a computer program do this task, we have to incorporate a reasonable amount of the knowledge into the knowledge base in a usable form. The nature of such knowledge, however, is vague and highly human-oriented. In an environment like this, the soundness and the flexiblity of the framework are extremely important. The framework for machine translation must be flexible enough to represent wide range of knowledge and to handle various exceptional cases uniformly. At the same time it must be sound in such a way that it can provide a designer with a clear and systematic guideline for refining vague notions into more precise and well-defined terms.

The basic notions we use to achieve these goals are active objects and functionality. Active objects are entities which can serve both as a data and as a procedure. We associate each lexical item of natural language with an appropriate active object. It not only contains declarative descriptions about the lexical item but it also embodies a procedure for manipulating descriptions of other lexical items. An active object for a lexical item specifies a word specific rule which can override the general procedure. This lexically driven feature provides a flexible and uniform way for dealing with exceptions in natural language. On the other hand, the principle of functionality constrains the interactions among active objects, preventing the whole system from falling into so-called complexity barrier. For example, it inhibits computation by side effects or that by opaque context. Montague grammar gives a basis for this treatment. We show how semantic interpretation is carried out based on the functionality principle.

The translation process based on this idea is divided into three stages: analysis, transfer and generation. In the analysis stage, an input sentence is analyzed both syntactically and semantically and lexical items of the input sentence are arranged into a function notation. This notation reflects semantic dependencies among lexical items in the sentence. Montague grammar provides both a basis and

a mechanical procedure for obtaining this analysis. In the transfer stage, each lexical item is replaced by an active object by consulting a bilingual dictionary, and in the generation stage, the resulting formula is evaluated. As a result, semantic interpretation is made in terms of the target language concepts, and at the same time a syntactic structure for referring to the resulting conceptual entity is created. Thus the language translation and semantic interpretation are carried out simultaneously.

We have constructed a prototype system for English-Japanese translation. Additional features such as heuristic rewriting rules for refining output Japanese sentences are incorporated into the prototype to make it more flexible. We have tested the prototype against sentences taken from scientific and engineering literature.

In this thesis we also explore basic issues related to machine translation.

First, we explore a procedure for extracting information from function notations we use as intermediate representations. From a computational point of view, it is important to argue how to reconstruct the memory structure based on the interpretation of the input sentence. Unfortunately, Montague grammar tells almost nothing about this question. To answer it, we use a semantic network formalism as a memory structure and elaborate a procedure for building or modifying semantic networks. We give detailed descriptions about the information interpretation procedure for several major syntactic constructions of English.

Second, we investigate the design issues of a parser for English. We put stress on making it easy to develope a large grammar. Although control issues must be taken into account in designing a large grammar, the grammar designer may not want to be involved in such issues in early stages of the development. Reflecting this aspect of grammar development, we have designed a grammar formalism consisting of two levels: augmented phrase structure grammar and procedural grammar. The procedural grammar is closely related to the augmented phrase structure grammar in such a way that the initial version of the procedural grammar can be obtained from augmented phrase structure rules only by slightly modifying the syntax. Then the procedural grammar can be further refined by incorporating a control structure. Interactive facilities and pattern directed dictionary management are also stressed.

Acknowledgements

I would like to express my sincere thanks to Professor Shuji Doshita of Kyoto University for the supervision and continuous encouragement to complete this thesis. He not only guided the author to the present study but also gave me many helpful suggestions in the course of this study. He also read the draft of this thesis with critique mind and gave me accurate comments, which were very effective for improving this thesis.

I would like to express sincere appreciatiation of accurate comments and insights given by Professor Makoto Nagao of Kyoto University.

I would like to appreciate accurate discussions made by Associate Professor Junichi Tsujii of Kyoto University.

My great debt is to the students who have worked with me. Mr. Masaki Kiyono did a good deal of work for writing an actual computer grammar for English. Mr. Akira Kosaka prepared a detailed documentation of the prototype system. He also did a work closely related to this thesis. Other present and past members of Prof. Doshita's laboratory have taken a time for discussion and provided many useful comments. These include Associate Professor Susumu Yamasaki, Dr. Shigeyoshi Kitazawa, Mr. Yuusaku Nakata, Mr. Youzou Sakakibara, Mr. Takao Yamanaka, Mr. Shigetsugu Zen, Mr. Tsuyoshi Morii, Miss Yiming Yang, Mr. Tomoo Yazaki, Mr. Tadashi Kawamura, and Miss Yumiko Kabeya.

Mr. Sidney J. Atkins of Indiana University read the draft of this thesis carefully, pointed out errors and rhetrical problems, and showed me precise and readable alternatives.

Finally, I cannot begin to express my appreciation to my parents, my wife Sanae, and my son Ryosuke for their support and encouragement.

Contents

1. Introduction

The ultimate goal of this thesis is to construct a computer program which translates sentences from one natural language into another. Until recently, language translation has been done only with highly sophisticated human expertise. In order to make the computer do this task, we first have to give a formal description both to the process of language translation and to the knowledge utilized in that process. The next step is to build a computer program which can simulate that process by referring to the encoded knowledge.

The most serious problem in building a machine translation system is that the machine translation program has to handle a wide range of natural language phenomena. Unfortunately, phenomena in natural language are so broad and complex that no linguistic theory developed so far can serve as a comprehensive framework which explains almost all natural language phenomena; any linguistic theory has to admit the existence of some exception which cannot be given a suitable answer by that theory. This is more serious in language translation, where we have to deal with the additional problem of meaning preserving translation.

Because of this, we cannot give a complete design in advance. At the implementation stage, we may encounter a problem which was not expected at the design stage. In such a situation, the engineer of a machine translation system has to give a tentative solution to the problem by making some heuristic considerations. This kind of trial-and-error feature is unavoidable in natural language processing in general. If the designer employs an "ad hoc and brute force" solution, his program would soon reach the "complexity barrier".

All we can do seems to be to employ strong guidelines to prevent the system from falling into complication, even if a number of heuristic solutions may be employed by a rule writer. The following two strategies are those employed by the software engineers to solve complexity barriers in software system design:

use of high level language, in which we can write compact and highly abstracted, problem-oriented instructions, and,

constraining the arbitrariness of programming: structured programming methodology is a good example.

These notions should also be taken into account by machine translation system engineers. More specifically, the following design criteria are considered to be important in designing a well-engineered machine translation system:

Richness of the Expressive Power of Formal Representation. In intermediate representation, various distinctions made in natural language should also be made. For example, the distinction between partial and total negation should be made.

Conciseness. A machine translation system should be compact enough to preserve readability and ease of understanding.

Simple Control Structure. This requirement is concerned with the dynamic structure of the system. The control structure of the system must be simple enough to allow the designer to gain a good perspective on how the system solves each problem in language translation.

Maximum Utility of Knowledge. We want to avoid the duplication of essentially the same knowledge, both to preserve integrity and to decrease the effort of incorporation of knowledge. For example, a word choice mechanism for a verb-noun phrase combination has to be applicable to all of its syntactic variations such as passive construction or relative clauses. The architecture of the system must allow the designer of the machine translation system to easily write rules which reflect the regularity of the language.

Exception Handling Capability. Exceptions have to be dealt with uniformly. In other words, the system needs to provide a mechanism for overriding general procedures by an exception handler without affecting unrelated parts of the system.

Extensibility. The grammar rules should be extended easily. Any ad hoc solution has to be easily replaceable by a more general solution.

In this thesis, we attempt to establish a model which satisfies the above criteria by concentrating on a specific language pair: English-Japanese translation. This

language pair is interesting both from a practical point of view and from a theoretical viewpoint.

Apparently, an essential component of a machine translation system depends on an underlying linguistic theory. Below we first examine the conventional frameworks and discuss their problems. Then, we will introduce a new linguistic theory, Montague grammar, which is considerd to be a good candidate for satisfying these criteria. Finally we will raise the questions asked in this thesis.

1.1 Conventional Framework of Machine Translation

Since the first apearance of computers, many researchers have been involved in building a machine translation system, and a number of experimental systems and commertial systems have been constructed[Hutchins 78], [Nagao 79a,b]. Early machine translation systems employed a brute force method and tried to translate sentences in a string-to-string manner. Apparently this did not work satisfactorily and deeper structures come to be put into use. The commonly used frameworks are phrase structure grammar and case grammar. So called *second generation systems* use such intermediate representations. For example, Metals[Lehmann 81],[Slocum 83] uses case structures, Ariane 78[Boitet 80] uses what they call multi-level trees, which is actually a mixture of phrase structure representaiton, case structure, and logical representation, etc. In English-Japanese or Japanese-English translation, such structured approaches have been most commonly used[Sakai 66,69],[Sugita 68],[Ishihara 74a,b],[Shudo 77],[Nagao 80,82],[Nishida(F.) 80,82],[Uchida 80],[Nitta 82]. At the same time, many techniques for natural language processing have been established[Nagao 77,78].

The question to be asked in this section is whether or not conventional frameworks are useful in the English-Japanese case. This section examines two approaches, that based on the phrase structure grammar and that based on the case grammar.

Machine Translation based on Phrase Structure Grammar

Phrase structure grammar mainly deals with syntactic aspects of language using the notion of a phrase. Based on this view, the language translation process can be modeled as a process of rewriting phrase structures of the source language into those of the target language. This is the simplest modeling of structured language translation. In fact early machine translation systems were built based on this idea.

However, phrase structures are heavily syntax-oriented and not useful for semantic processing, since semantic relationships among constituents could not be expressed directly with phrase structure trees. For example, the deep object of a transitive verb of English can be placed at various positions of a phrase structure tree, depending on the voice or other factors of a sentence. Such a characterization is heavily language dependent. If we were to design a machine translation system based on such a viewpoint, we had to incorporate a large number of structural transformation rules, which would make the system unthinkably complex.

What is needed is, therefore, some sort of canonical form for representing semantics. One possibility in solving this difficulty within phrase structure paradigm might be to augment phrase structures by annotating each node. But such solution is not very useful in descreasing the complexity of the translation procedure when the distance between the source language and the target language is large.

Machine Translation based on Case Grammar

With case grammar, we can use case structures to give a canonical representation to a variety of syntactic structures which can be thought of as having the same meaning. Furthermore, case structures have been thought of as language independent. In designing a machine translation system, the notion of cases and semantic markers gives a clue for word choice and meaning oriented translation.

However, one problem with case structures is that their expressive power is not sufficient; for example, it seems to be difficult to represent the difference of scope to make a distinction between partial and total negation. Although it could be possible to augment case structures to handle this phenomenon, the resulting representation would be awkward. Instead, it would be better to treat such issues in a different level of representation.

The second problem is that the notion of case is still unstable; there have been lots of controversies as to the identification of a universal set of case labels or semantic markers. As a result, there has not yet been established any commonly accepted procedure for determining deep cases from surface structures of language. So it would be dangerous to design a machine translation system by heavily depending on case grammar.

1.2 Montague Grammar

In the early 70's, a new paradigm of linguistic theory called Montague grammar was proposed[Montague 74a,b]. One of the most significant features of Montague grammar is in its semantic component. Montague grammar attempts to treat the semantics of natural language in the same ways that logicians analyze the semantics of logic.

The semantics of natural language are characterized by a series of strictly defined mappings from the natural language domain to the mathematical domain. According to Dowty (78) and (81), Montague grammar is characterized by the following features:

Truth Conditional Semantics. This means that *to know the meaning of a (declarative) sentence is to know what the world would have to be like for the sentence to be true.*

Model Theoretical Semantics. This means that the framework of Montague grammar involves *construction of abstract mathematical models of those things in the world making up the semantic values of expressions in the object language.* In other words, this corresponds to a view of *linguistics as mathematics.*

Possible World Semantics. This correponds to the idea that *the meaning of a sentence depends not just on the world as it in fact is, but on the world as it might be, or might have been, etc.*

From a computational point of view, the novelty of Montague grammar is that:

the framework is based on formal semantics,

a procedure for mapping natural language expressions into semantic domain is explicitly shown,

semantic mapping is defined based on the principle of compositionality.

Those characteristics of Montague grammar deserve careful study.

1.3 Questions

In this thesis, we raise the following question to be answered:

Could this formal treatment of natural language semantics be useful in actual machine translation?

Computationally, we have to raise the second question as follows:

How can we build a machine translation system based on Montague grammar?

In this thesis, we attempt to answer these two questions empirically, by constructing an experimental system which can demonstrate features of the approach.

2. Overview of the Machine Translation System

2.1 Introduction to Chapter 2

This chapter overviews our machine translation system, trying to single out the major ideas introduced in this thesis.

Use of Logical Expressions as Intermediate Representation

Our machine translation system is characterized by the use of logical expressions as an intermediate language. We call this logical language EFR, or English-oriented Formal Representation.

EFR provides a strict syntax for describing unambiguous reading of English phrases. For sentences which are thought of as having ambiguities, more than one expression in EFR is assigned.

In EFR, the role of each word in an input sentence is represented in function-argument pattern. If word A modifies another word B to give additional information, we think of A as a function to B and use the notation:

A(B).

For example, we can think of an adjective as a function taking the modified noun as an argument, since an adjective modifies a noun to give attributive information to the noun. Thus we could write:

big(apple).

Usually, an adjective may in turn be modified by an adverb, as:

surprisingly(big).

Thus, EFR is a higher order language in its formalism.

In order to make this function-argument analysis for each natural language expression, we need a basis for semantic analysis. Montague grammar offers

suggestions for dealing with this problem. As in Montague grammar, for example, we analyze noun phrases as functions to verb phrases. Assignment of EFR expressions to each phrase structure of English is done based on semantic considerations, as described in chapter 3.

Annotated Phrase Structure for Describing the Syntactic Structure of the Target Language

Another kind of formal language is called CPS or Conceptual Phrase Structure. It is used to provide a description of Japanese syntax structure. CPS is an annotated phrase structure each node of which has semantic information in attribute-value form.

Translation as Functional Computation

Operations on CPS are tree transformations including tree synthesis and feature calculation. CPSF or CPS Function is a language for denoting operations on CPS. Lambda notation is used to denote functional abstraction.

Our machine translation system operates in a sentence-by-sentence manner. The overall translation process is defined as a three staged process:

Analysis Stage

In this stage, we analyze English sentences and extract expressions of EFR. Conceptually, a parse tree is translated into an expression of EFR by making syntax directed translation. Actually, syntactic analysis and semantic composition are done in parallel, and a number of features are used to check various agreements.

Transfer Stage

In this stage, we substitute formulas of CPSF to each lexical item in the extracted expression. Thus our bilingual dictionary is a set of lexical-item and target language generating function pairs. This makes our machine translation system lexically driven. We can substitute an arbitrarily complex CPSF function into each lexical item, if the type of function is agreed on. This makes it possible to translate an English lexical item into composite structure in Japanese. In the transfer stage, however, we cannot write a rule for reducing several English words into one Japanese lexical item. Such a rule can be given either as an analysis rule

like those dealing with idiomatic expressions or as a heuristic rewriting rule which will be applied after the Japanese syntax structures are generated.

Generation Stage

In this stage, we evaluate the formula resulting from the transfer stage by viewing it as a function for generating target language structure. The result is a structure of CPS. Before making morphological synthesis, we need to eliminate redundant expressions if any exists. Such expressions may arise out of local processings of CPSF evaluation. We use a small set of heuristic tree-tree transformation rules to replace such expressions by more suitable ones.

Figure 2.1 gives a rough sketch of the translation process.

2.2 Translating English into Formal Representation

The simplest and most natural way to translate a given phrase structure into expression of EFR would be to give the two kinds of association rules:

(a) lexical rule: this type of rule states how each lexical item corresponds to expression of EFR.

(b) composition rule: for each syntactic derivation rule, this type of rule specifies how to get an EFR for a parent node from EFR's for its descendants.

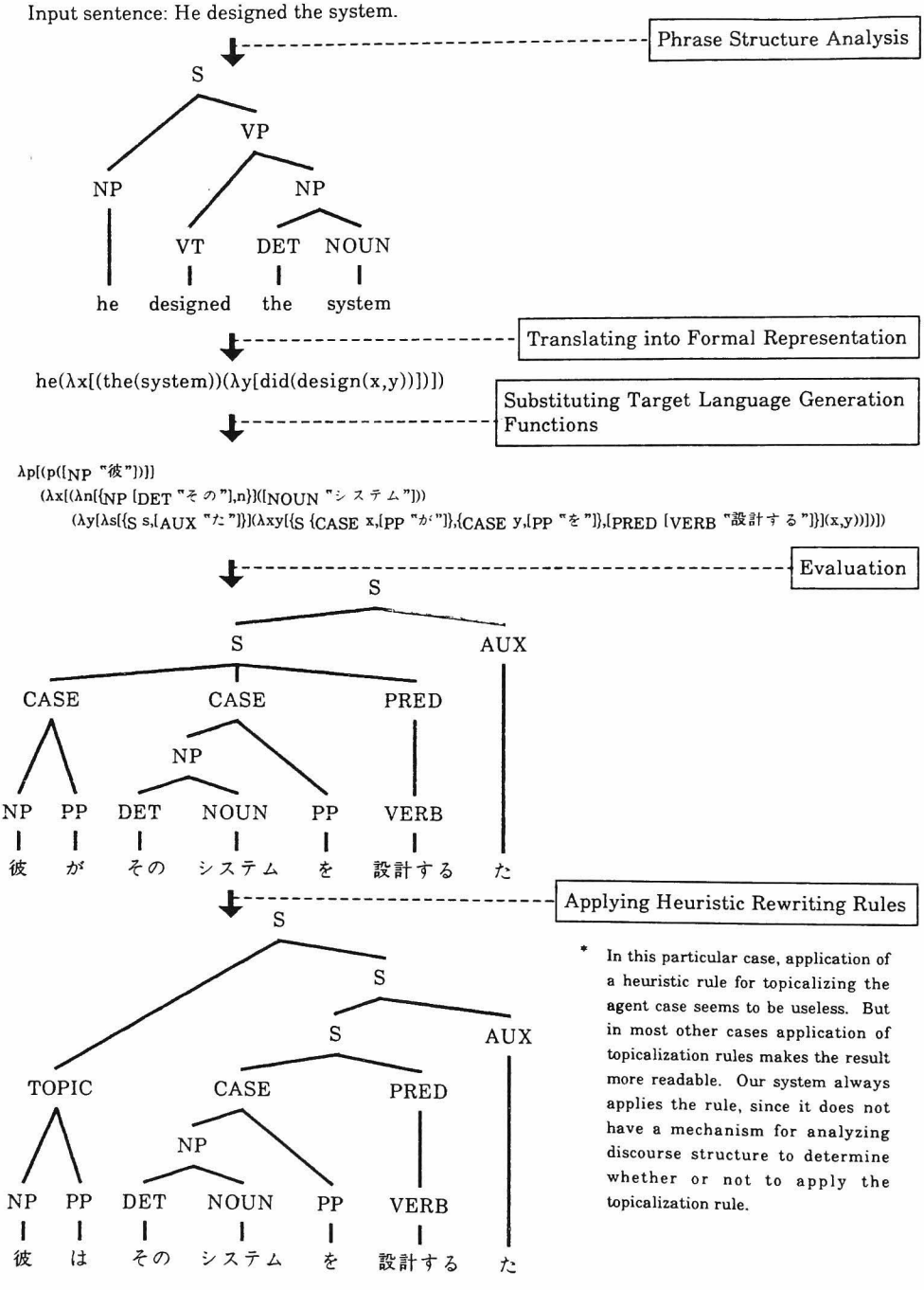
(Example)

NP → DET.NOUN where <NP> = <DET>(<NOUN>).

This rule specifies that an NP consists of a DET and a NOUN, and that the EFR expression for the NP is a function form with the EFR expression for the DET as a functor and that for the NOUN as its argument.

In figure 2.2, we illustrate a number of other sample rules describing the syntax and semantics of a small fragment of English.

Referring to a set of rules defined in this way, we can obtain a logical form of EFR for an input sentence in a step-by-step manner, as illustrated in figure 2.3. Each node of trees in figure 2.3 represents a pair of a syntactic category and an EFR expression for the node.



* In this particular case, application of a heuristic rule for topicalizing the agent case seems to be useless. But in most other cases application of topicalization rules makes the result more readable. Our system always applies the rule, since it does not have a mechanism for analyzing discourse structure to determine whether or not to apply the topicalization rule.

Figure 2.1 Sketch Description of Translation Process.

$S \rightarrow NP.VP$ where $<S> = <NP>(<VP>)$

$VP \rightarrow VT.NP$ where $<VP> = \lambda x[<NP>(\lambda y[<VT>(x,y)])]$

$NP \rightarrow DET.NOUN$ where $<NP> = <DET>(<NOUN>)$

$NOUN \rightarrow ADJP.NOUN$ where $<NOUN> = <ADJP>(<NOUN>)$

$NOUN \rightarrow NOUN.+S$ where $<NOUN> = {}^*pl(<NOUN>)$

*pl is a special symbol standing for plural.

$NOUN \rightarrow NOUN.PP$ where $<NOUN> = <PP>(<NOUN>)$

This analysis depends on Montague's analysis. In writing an actual grammar for parsing, we use an ordinary rule: NP → NP.PP rather than the above rule, for the above rule is a source of ambiguity. A special kind of semantic composition rule is used in that case.

$PP \rightarrow PREP.NP$ where $<PP> = \lambda p[\lambda y[<NP>(\lambda x[((<PREP>(x))(p))(y))])]]$

$VP \rightarrow BE.(VT)_{PASSIVE}$ where $<VP> = {}^*en(<VT>)$

*en is a special symbol for changing the type of its argument from a two place function to a one place function by supplying a dummy argument.

$VP \rightarrow VP.PP$ where $<VP> = <PP>(<VP>)$

Figure 2.2 Example of Rules for Making English-EFR association.

As is understood from the example in figure 2.3, the final annotated phrase structure suffices for seeing all information of intermediate stages. Figure 2.4 illustrates other examples of English-EFR association.

Augmented Phrase Structure Grammar for English

In writing an actual grammar for English, however, the phrase structure grammar defined above does not have sufficient power, so it needs to be extended. We make extension in two ways: (a) incorporating syntactic and semantic features for checking agreements among related phrases, and (b) extending nonterminal symbols so as to denote structures from which some phrase is moved away (e.g., relative clauses can be defined as sentence structures with exactly one noun phrase moved outside). Figure 2.5 and 2.6 illustrates examples of those extensions.

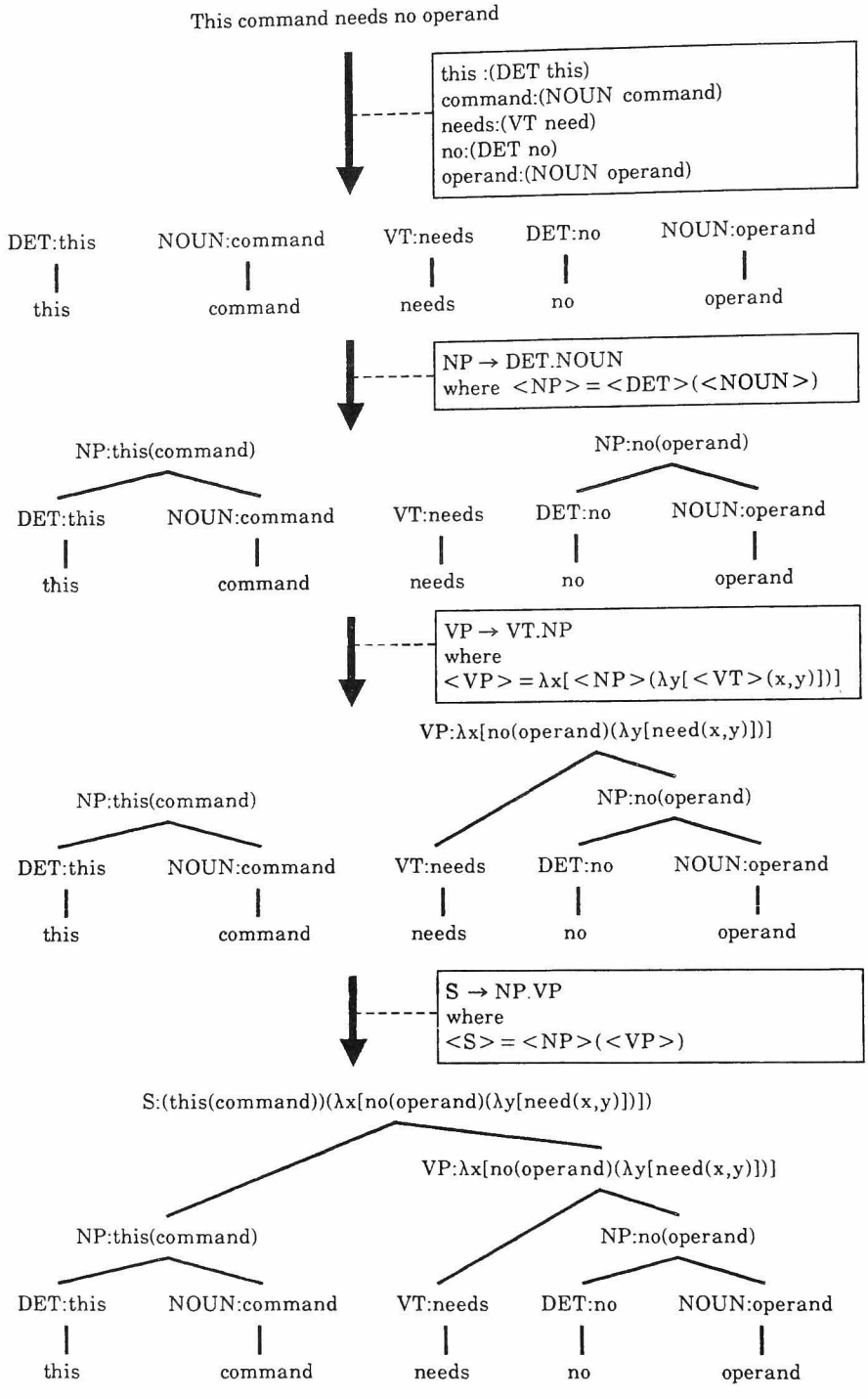
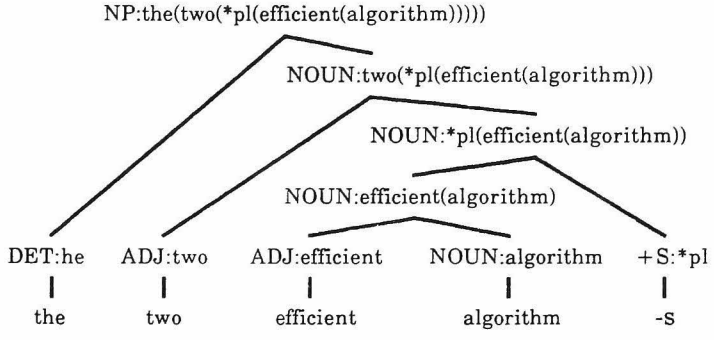
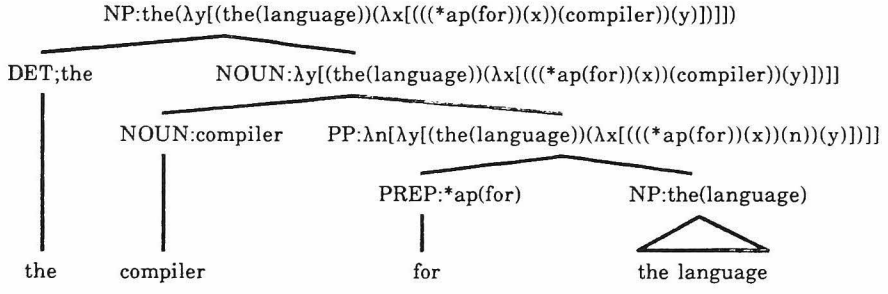


Figure 2.3 Assigning an Expression of EFR to an Input Sentence.

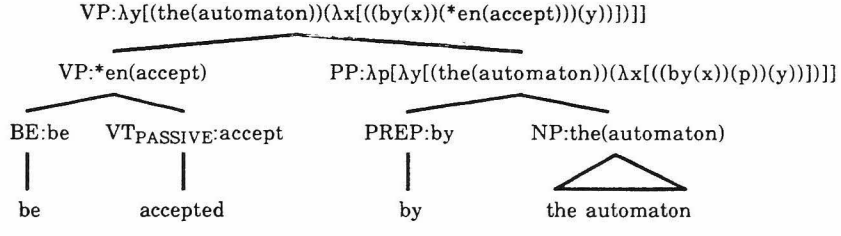


(a) NP: "the two efficient algorithms"



In actual grammar for parsing, the structure is somewhat different from this structure (NP → NP.PP rule is used). But this is only for convention and the resulting expression in EFR is the same.

(b) NP: "the compiler for the language"



Conceptually, we assign two expressions in EFR to each preposition: by ←by, and by ←*ap(by). The former is used for adverbial PP while the latter is for adjectival PP. In the actual grammar for parsing, this distinction is made later to decrease ambiguity.

(c) VP: "be accepted by the automaton"

Figure 2.4 Examples of English-EFR Associations.

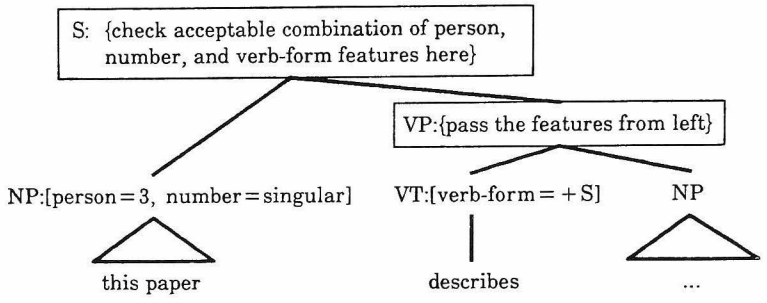


Figure 2.5 Subject-Verb Agreement Checking Using Features and Test.

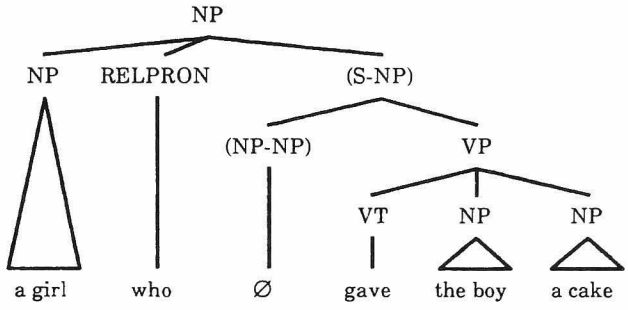
Translating Declarative Rule into Procedural Rules

In actual parsing by a computer, it would be inefficient to use these kinds of declarative rules and dictionary directly. It would be more efficient if we attach additional informaiton to each rule, to explicitly tell the parser when to invoke the rule.

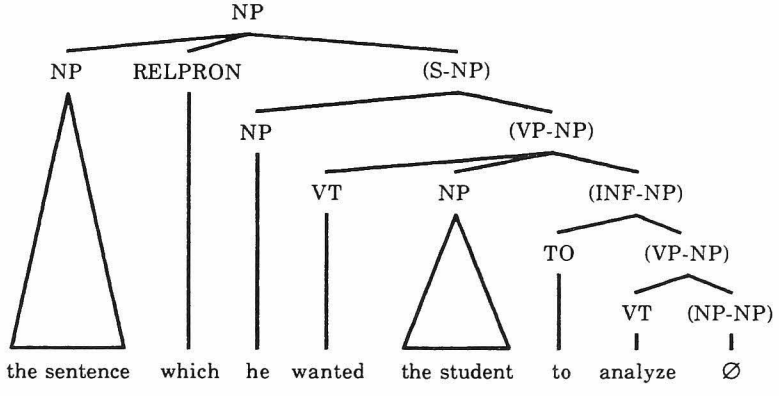
The parser uses procedural rules. There are four types of procedural rules: E-, U-, B-, and L-rules. An E-rule will be invoked when an associated goal is attempted by the rule interpreter. A U-rule will be invoked when a specified partial tree is built by the rule interpreter. A B-rule is applied in a bottom-up manner. Unlike the other three types of nodes, L-rules are embedded in the dictionary and will be invoked only when the lexical item appears in the input sentence.

Dealing with Ambiguities

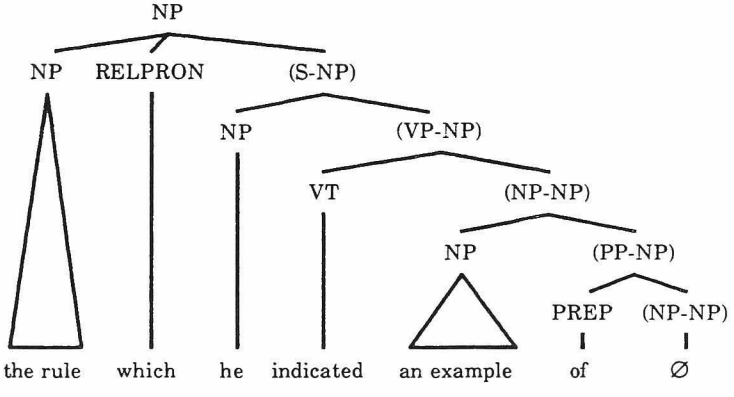
One of the serious problems in parsing natural language is the treatment of ambiguity. There are sentences which are syntactically well-formed but semantically not. Most current natural language systems use a number of semantic markers to solve this problem. However, this approach seems to be only useful in a restricted domain where domain specific markers precise enough to resolve ambiguities are defined. Even so, there is a possibility that ambiguity might not be resolved.



(a) "a girl who gave the boy a cake"



(b) "the sentence which he wanted the student to analyze"



(c) "the rule which he indicated an example of"

Figure 2.6 Augmenting Phrase Structure Grammar for Dealing with Holes.

We introduce a notation (α-β) to stand for a structure labeled α that has a hole for β somewhere within it. We can write a rule:

Rule: NP → NP.RELPRON.(S-NP) where ... .

to give structural analyses as the above.

Hence, the parser needs to have a mechanism to cope with ambiguities.

There are two approaches to this problem: one is to obtain all possible analysis at the same time by bottom-up parallel algorithms, and another is to output the candidates one-by-one using some kind of backtracking algorithm. We take the latter approach mainly because of the limitation of memory space.

Interactive Diagnosis

The problem arising in the latter case is that the correct answer might be located at the last position of the candidate list the system produces. At worst, we might have to repeat the examine and reject cycle many times.

Our parser has a mechanism which allows the user to tell which part of the analysis is wrong and which is right. It provides an interactive facility to look into a parse tree produced by the parser. If such diagnostic information is given, the parser will try to find the next candidate within the constraint given by the human superviser. Of course, the user can simply reject the result and make the parser search for the next candidate. Figure 2.7 illustrates an example of man-machine interactions. This mechanism is useful in decreasing the number of interaction.

Another possibility is to use heuristics to give the plausible analysis high priority and to make the parser output the most plausible analysis first. In our parser, however, this mechanism is difficult to incorporate. This problem is related to the completeness of the parser and will be discussed in chapter 4.

Utilities for Dictionary Management

Dictionary management is important in making the experimental system design effective. Even in the late stage of designing a grammar, we might want to change the specification of the dictionary format as well as the content. A pattern-directed dictionary manager is developed to make the design improvement easy.

2.3 Generating Target Language from Formal Representation

We can generate target language expressions just as we interpret the logical form word-by-word. We use an augmented phrase structure called CPS to denote the result of the interpretation. We use a distinguishing name of CPS (Conceptual Phrase Structure) to put emphasis on the conceptual role played by CPS.

CPS -- Conceptual Phrase Structure

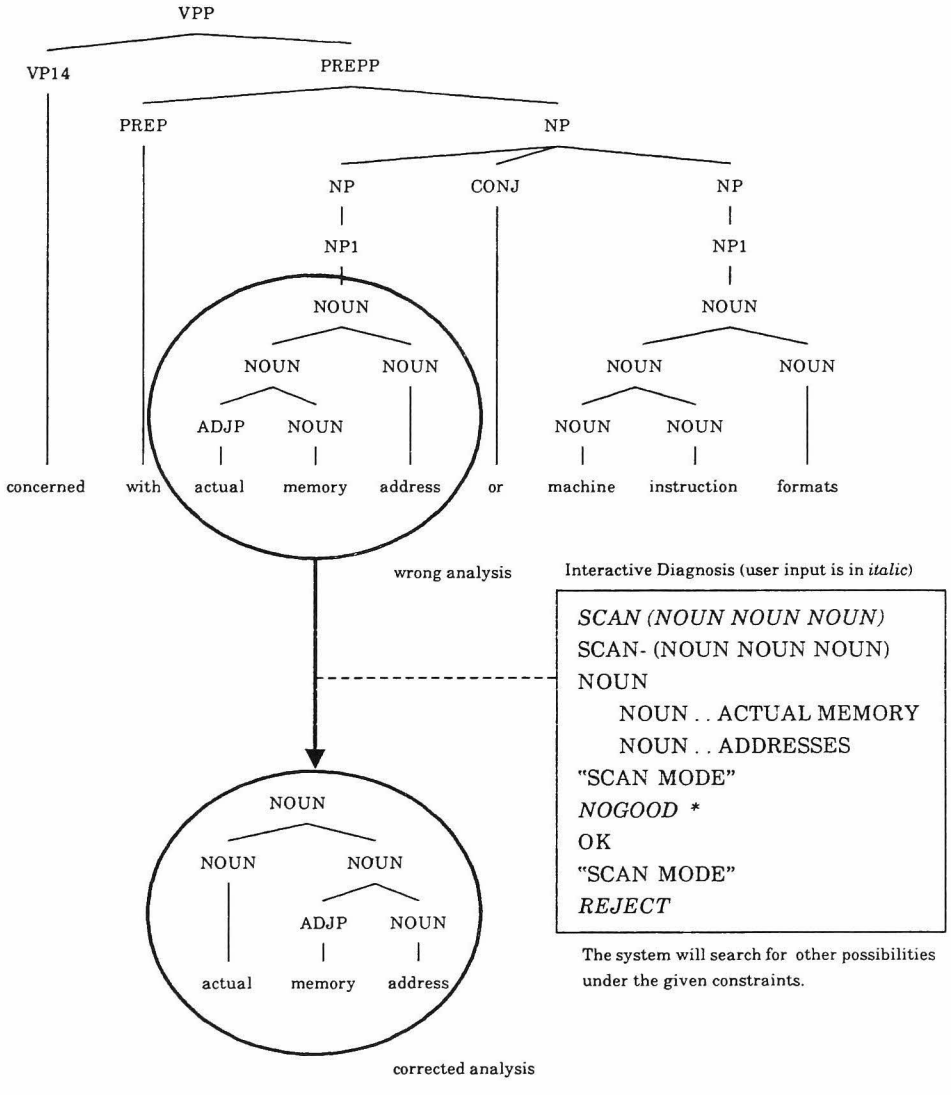


Figure 2.7 Example of Interactive Diagnosis.

The structure of CPS is a tree structure with each node annotated by a set of attribute-value pairs. Each attribute represents a semantic or syntactic feature for augmenting the information conveyed by each node of the tree.

(Example) a noun "猫"(cat)

[NOUN "猫" with class=animate, count=able, ...]

Sometimes we use graphical notation as shown in figure 2.8 to help the reader comprehend the structure of CPS structures easily:

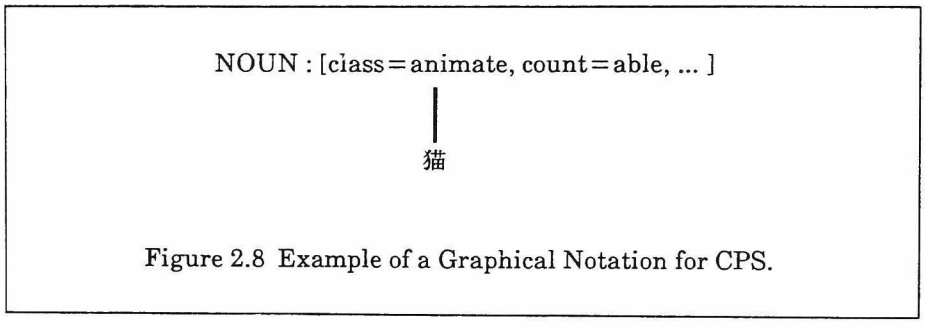


Figure 2.8 Example of a Graphical Notation for CPS.

CPSF -- Functional Language on CPS

Operations on CPS are represented in a functional language CPSF. Structures in CPS are involved in CPSF as constants. Other constituents of CPSF are: variables, composition, tree transformation, functional application, and functional abstraction using lambda notations. A LISP-like interpreter is defined to evaluate formulas of CPSF.

(Example 1) tree transformation

$$+\text{TENSE}_{\text{TENSE=PAST}}$$

This form represents an operation of transforming a given sentence into past form. +TENSE denotes a function name and the suffix TENSE=PAST specifies its parameter. The argument should be of category S or sentence. Figure 2.9 illustrates an example using this transformation.

Roughly speaking, we can define the above transformation using the following lambda form:

$$\lambda x[\{S\ x, [\text{AUX "た"}]\}],$$

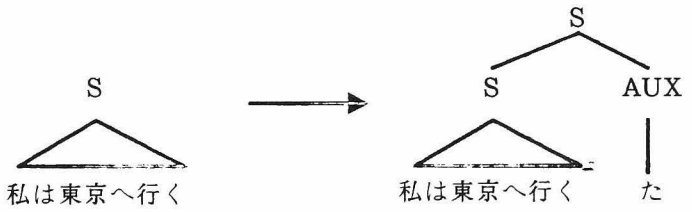


Figure 2.9 Transformation Made by a $+\text{TENSE}_{\text{TENSE}=\text{PAST}}$ operation.

where the form {CAT x,y, ... } will build a CPS of category CAT from x, y, and etc. (composition).

(Example 2) conditional form

λxy[{class(x)=animate → {S {CASE x, [PP "が"]},
{CASE y, [PP "を"]},
[VERB "壊す"]};
else → {S {CASE x, [PP "のせいで"]},
{CASE y, [PP "が"]},
[VERB "壊れる"]}}]

This form will generate different CPS's according to the semantic class of the first argument. This mechanism can be used for making word choice.

In the second example posed above, we may have the question whether the same rule can be applicable to passives or relative clauses in which the deep case is not explicitly placed in corresponding surface position. In our system, the answer to this question is positive. But the treatment depends on details of the CPSF, so we will explain it in chapter 5.

Generation as Evaluating Functions

Generation of the target language is done by evaluating a CPSF formula resulting from substituting an appropriate piece of formula of CPSF into each lexical item in a given expression in EFR.

Word Choice Problem

Using the feature complex associated with each node of CPS and conditional expressions, we can choose an appropriate expression of Japanese based on the meaning. An example is shown below:

(example)

input sentence: "The storm breaks it."

expression in EFR: (the(storm))(λx[it(λy[break(x,y)])

substitution: break ← λxy[{animate(x) = + → {S {CASE x, [PP "が"]},
{CASE y, [PP "を"]},
[PRED [VERB "壊す"]]};
else → {S {CASE x, [PP "のせいで"]},
{CASE y, [PP "が"]},
[PRED [VERB "壊れる"]]}}]

it ← λp[P([NP "それ"])]

the ← λnoun[λp[p([NP [DET "その"], noun])]]

storm ← [NOUN "嵐" with animate = -]

Evaluating a formula resulting from the substitution will result in a CPS structure:

[S [CASE [NP [DET "その"] [NOUN "嵐"]][PP"のせいで"]]
[CASE [NP "それ"][PP "が"]]
[PRED [VERB "壊れる"]]],

The equivalent graphic notation is shown in figure 2.10.

If the input sentence is:

"He breaks it",

then a different verb will be used in the resulting sentence. (See figure 2.11).

Actually, most other systems attempt to make word choice or structural transfer by elaborating their own semantic markers. But no complete answer has been obtained as to how many markers and what kinds of tests are sufficient for making the appropriate word choice or structural transfer. Such questions can only be

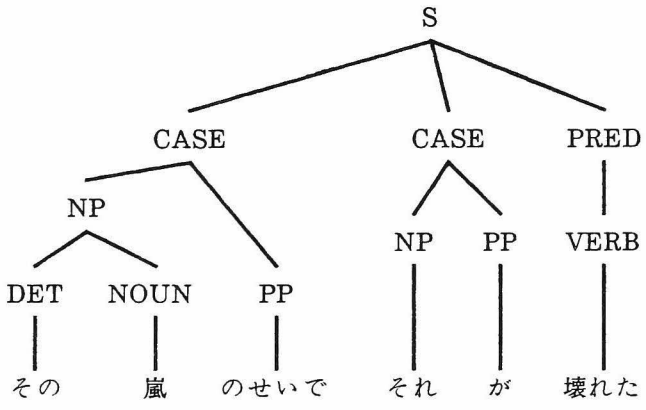


Figure 2.10 Graphic Notation of a CPS structure Generated from the Sentence: "The storm breaks it."

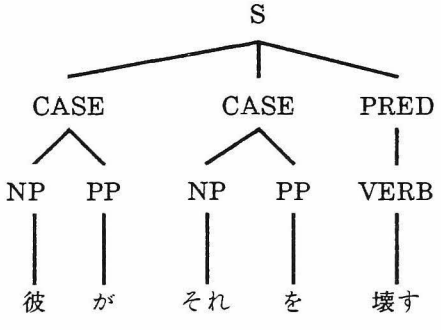


Figure 2.11 Graphic Notation of CPS Structure Generated from the Sentence: "He breaks it."

answered through comprehensive study of natural language phenomena, but answering these questions is beyond the scope of this dissertation.

In this dissertation, we use a very simple set of features such as:

abstract(±), action(±), animate(±),number(±), count(±), and ako(<superclass>)

and we will discuss what we call the feature passing problem: for example, we will discuss a mechanism for making a feature complex of an object case NP accessible from the verb even if the relation is implicit in the surface level as in passive sentences or relative clauses.

Higher Order Problem

Generation of structures of CPS is done by evaluating formulas of CPSF. Since formulas of CPSF are higher order in their formalism, a problem called the *higher order problem* arises.

The higher order property of EFR is inherent in formulas of CPSF. This poses what we call the higher order problem in evaluating formulas of CPSF. The problem is how to deal with modifications to functions.

For example, the straightforward assignment of a CPSF formula to an English adjective is a function which takes a noun to produce a complex noun structure by attaching the corresponding Japanese adjective to the given noun. For instance,

large ← λnoun[{NOUN [ADJ "大きな"], noun}].

Imagine a case where that adjective is modified by an adverb, say "very". In that case, a CPSF function assigned the lexical item "very" must be something which will transform the above lambda form into:

λnoun[{NOUN [ADJ [ADV "とても"][ADJ "大きな"]], noun}].

Note that we cannot assume the *form* of a given lambda form. All we can know is the *type* of a given lambda form. Consequently, the above task requires analysis of the given lambda form as well as determining the output form. Since this task is too complicated, we use heutistic solutions. For example, we use the notion of the application rule. It is based on an object-oriented view. A CPS structure is viewed as both data and function. Figure 2.12 illustrates how we use this notion in generating target language structures.

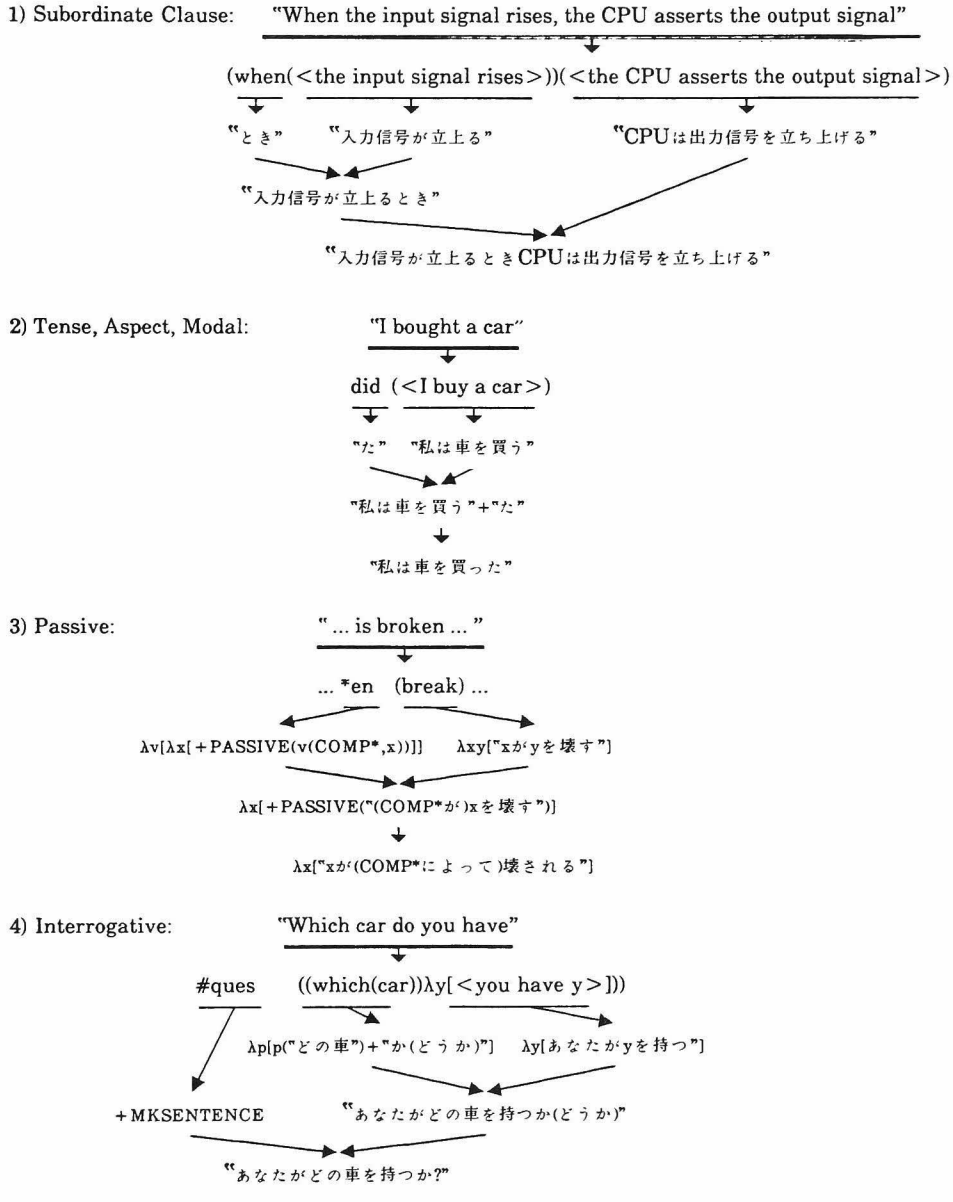


Figure 2.12 Generating the Syntactic Structure in Japanese.

In chapter 5, we will describe the treatment of this problem in detail.

Use of Heuristic Rules to Reform Output

Generally, in the generation stage, we have to be careful enough to prevent ambiguous, lengthy, or illegible expressions from being generated. Although we could incorporate into each function a mechanism for detecting those expressions and replacing them by appropriate expressions, it would complicate each generation function and spoil the simplicity of the generation stage. Accordingly, we have a separate stage called a "reform" stage for carrying out this process. The reform stage is applied after rough output is obtained in the generation stage.

A set of heuristic tree-tree transformation rules are utilized in the reform stage. Figure 2.13 illustrates an example.

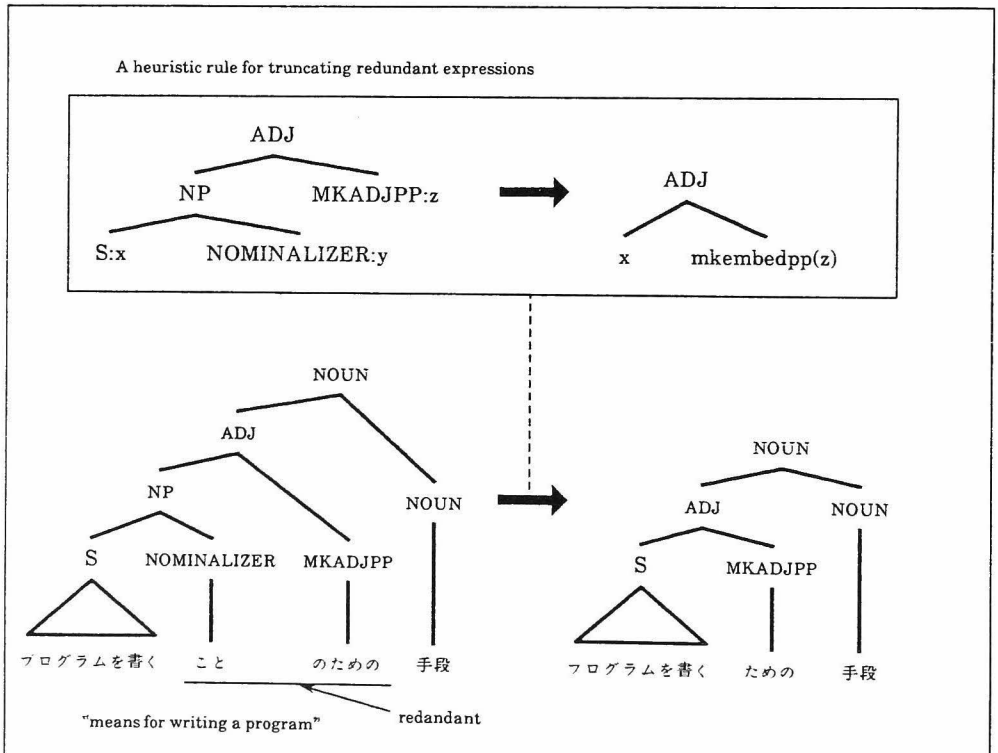


Figure 2.13 Example of a Rule Used in the Reforming Stage.

As heuristic rewriting rules are defined almost independent of the main flow of translation, they are easily defined, modified, or deleted without affecting other parts of the system. The notion of heuristic rewriting provides the designer with a convenient means for giving a tentative solution to a problem whose treatment is really difficult.

Morphological Synthesis

The final stage is morphological synthesis. In this stage, leaf nodes are taken from the phrase structure tree of Japanese and inflectional processings are carried out for adjacent constituents. A notion of terminal string frame is introduced to deal with a number of exceptional cases of inflectional suffixes of Japanese.

3. A Semantic Network Model for Natural Language Analysis

3.1 Introduction to chapter 3

The distinguishing feature of our machine translation system is the use of logical language EFR as an intermediate language. This chapter describes how we associate logical expressions of EFR with each phrase of English. The principle behind this association is based on what we call a logical model of English. The basic idea of the logical model is indebted to Montague grammar[Montague 74a,b].

Montague grammar is characterized by model theoretic, truth conditional, and possible world semantics. Montague grammar uses a notion of "point of reference" to refer to the situation or context in which a given sentence is uttered. Imagine a sentence:

Yesterday, you gave me an apple. ... (1)

The truth value of this sentence depends on who utters this sentence, who is the hearer, or when it is uttered. The point of reference is an abstracted device to refer to such indices of the world and context. Montague grammar provides an explicit procedure for determining whether a given sentence is true or false in a given "point of reference".

According to Montague grammar, the "meaning" of a sentence is defined as a pair consisting of a set of contextual parameters and a set of points of reference each of which makes the sentence true. The "sense" of the sentence is defined as a set of world coordinates which make the sentence true once the context of the utterance (e.g., who is the speaker, who is the hearer, etc.) is fixed. Thus the meaning of the sentence is a function which takes the context of utterance and which will produce the sense of the sentence in that context.

From a computational point of view, this sort of treatment does not make sense unless the theory provides a finite means for referring to any set of point of

reference. Unfortunately, Montague grammar does not provide us with any such means. Some researchers in computer science have attempted to use intensional logic directly, but deduction within intensional logic is far from realistic amount of computation. This is a serious disadvantage of Montgaue semantics when we try to build a computer program based on the theory.

As contrasted with truth conditional semantics of Montague grammar, we use semantic network formalism to make a partial semantic interpretation. The semantic network structure we produce from a given sentence represents the typical situation implied by the sentence. In particular, referent resolution is made with previously generated semantic networks as a discourse structure. In this processing, mainly language driven inferences are made to replace intensional expressions by their referents. This is not only considered to be a reasonable processing by a language analyzer but it also significantly simplify the resulting network structure. This aspect should be distinguished from other systems which simply translate syntactic structures into logical expressions.

It has often been argued that the intensional logic component of Montague grammar is redundant and could be omitted[Dowty 81]. From a computational point of view, however, this discussion seems to be misleading. In this respect, we agree with discussion by Kaplan (1982) that it is important to have computationally meaningful intermediate representations in translating natural language expression into semantic representaiton. Intensional logic should be an important milestone from natural language to semantic representation. Our semantic model involves logical language as an important stage of semantic interpretation.

In summary, we illustrate in figure 3.1 the framework of our semantic model for natural language. An input sentence is analyzed and translated into an expression in EFR. Then a semantic network representing "new information" is produced.

In what follows, we will describe semantic network formalism and logical language EFR and then we will discuss the procedure for producing the semantic network from natural language expressions.

3.2 Semantic Network

A semantic network consists of nodes and links. Each node represents a concept and each link represents a relationship among concepts. The set of concepts

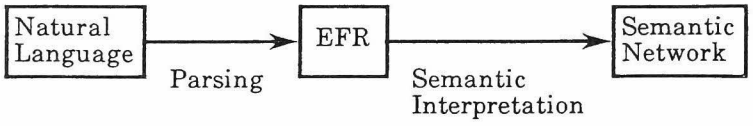


Figure 3.1 Framework of Semantic Model for Natural Language.

involve various entities: individuals, propositions, properties, relations, actions, and other thinkable entities.

3.2.1 Nodes and Links

Nodes are classified into class nodes and instance nodes. Class nodes convey stereotypical knowledge about concepts, while instance nodes stand for each occurence of a concept. In graphical notations, class nodes are represented in ellipses, while instance nodes are in circles. Instance nodes represent specific instantiations of a class node.

Links are denoted as directed arcs with labels. Links represent semantic relationship with other nodes. Typical labels attached to links are C(Class) and SS (SuperSet). C links are attached to instance nodes to identify their class nodes. On the other hand, SS links can be attached to each class node, indicating its *superclass*. If A is a superclass of B, then all instances of class B also have properties inherited from class A. Other labels attached to each link are deep case labels such as subject, object, destination, etc. The notion of case is inherited from case grammar. Figure 3.2 illustrates an example.

Note that we use nodes to represent abstract concepts (e.g., "possess") as well as concrete objects (e.g., "books"). Figure 3.3 illustrates examples of a semantic network representing an event.

We extend this notation to represent an event sequence (figure 3.4).

Figure 3.5 illustrates a semantic network representing causal relationship among events.

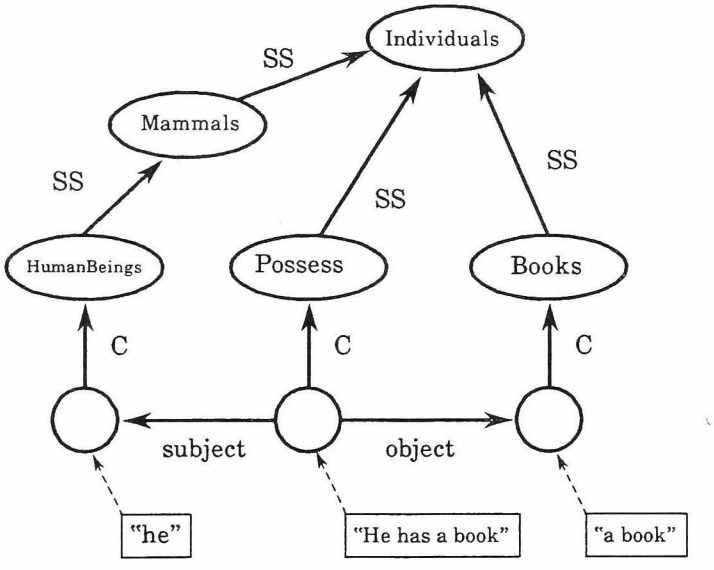


Figure 3.2 Example of a Semanticc Network, corresponding to "he has a book."

: class node : instance node

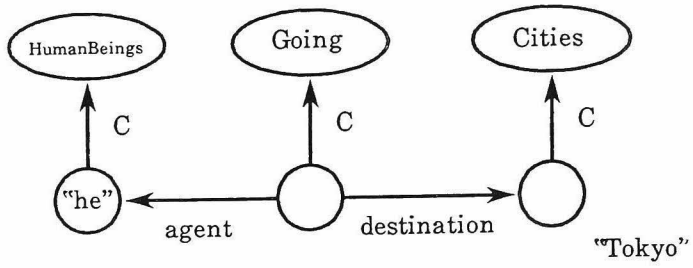
3.2.2 Linear Notation

We can use linear notations in place of structured notations. Actually we use relational representations similar to those used in Nilsson (80). For example, for the semantic network in figure 3.2, we use a set of linear notations as follows:

Exsists {N1, N2, N3}

C(N1, HumanBeings)
C(N2, Possess)
C(N3, Books)
SS(HumanBeings, Mammals)
subject(N2, N1)
object(N2, N3).

3.2.3 Paragraphs



(Past tense is neglected in this figure.)

" he went to Tokyo."

Figure 3.3 Example of Semantic Networks for Events.

We introduce a notation of situational descriptions called *paragraphs* to represent multiple-world semantics. The notion of paragraph is based on the partitioning of semantic network proposed by Fikes (77) and Walker (78). It is also used to denote belief context or other logically possible situations. A paragraph is depicted as a box. Figure 3.6 illustrates an example of the use of a paragraph for representing a past situation.

We can also use paragraphs to represent sentential relations of natural language. Figure 3.7 illustrates a semantic network structure representing a reason.

3.2.4 Special Representations

We introduce a number of special structures to make the translation from natural language easy. Introduced structures are: indefinite node, definite node, set node, and quantification. These structures are used to represent things intensionally. We first introduce a notion of a variable node and then explain other new structures.

3.2.4.1 Variable Nodes

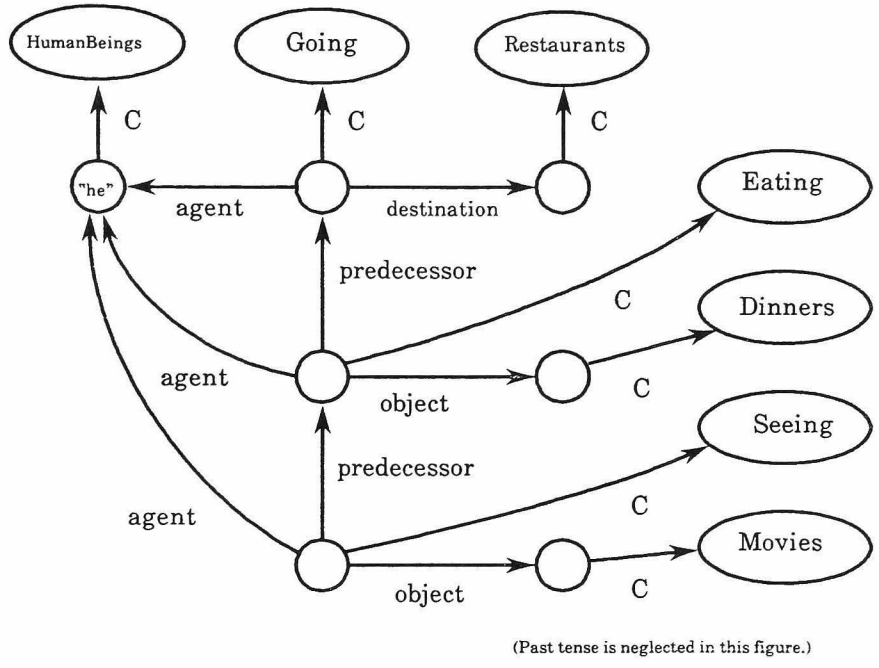


Figure 3.4 Example of Semantic Network for Event Sequence:
"he went to the restaurant, ate dinner, and saw a movie."

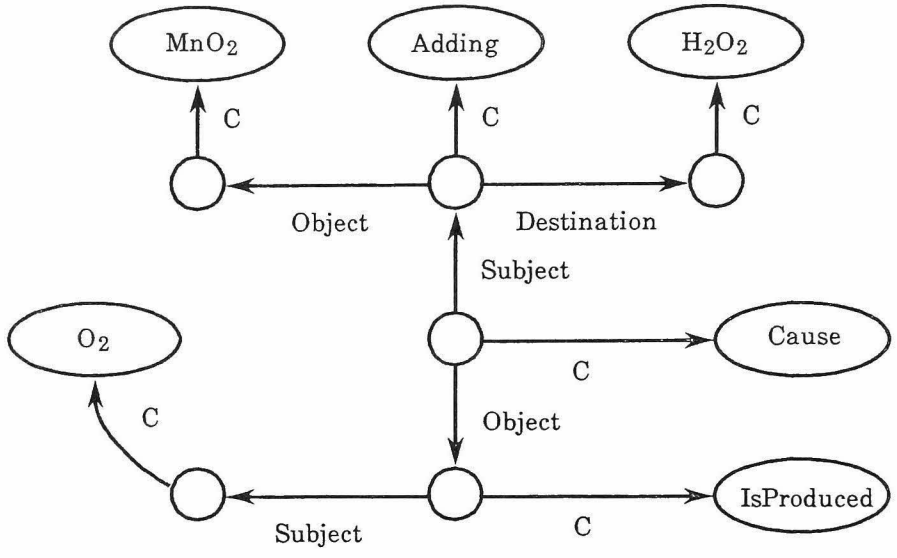
All of these intensional structures use variable nodes. In notation, variable nodes are denoted by a question mark followed by a variable name. For example,

?X, ?Y, ...

3.2.4.2 Indefinite and Definite Nodes

Figure 3.8 shows the structure of indefinite and definite nodes, together with linear notations for them. In figure 3.8, a couple of new links are introduced: VAR(iable) link and ST(such that) link. They represent bound variable and bound proposition, respectively.

These notations are introduced to deal with intensional expressions in natural language. By intensional expressions we mean those referring to objects whose



"O_2 was produced as a result of adding MnO_2 to H_2O_2."

Figure 3.5 Example of Semantic Network for Causal Relationship.

existence is not assured. We use such expression when we want to refer objects by their property. For example, a sentence:

(a) "she is building an efficient parser",

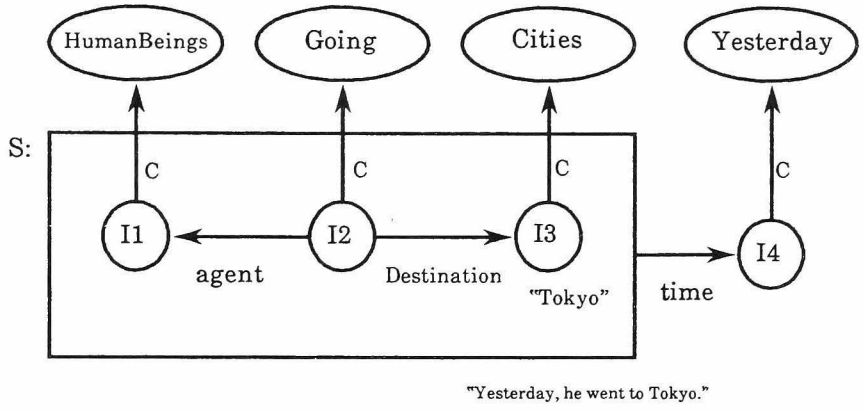
does not presuppose the existence of the "parser" at the moment when the sentence is uttered. The sentence only describes the property of the program she had in her mind. Accordingly, the semantic network structure for this sentence must be distinguished from that for sentences:

(b) "she has built an efficient parser."

(c) "she has built a parser which is efficient,"

(d) "she has built a parser, which is efficient,"

(e) "she has built a parser. It is efficient."



(a) Graphical Notation.

At the current context:

exists{I4, S}

C(I4,Yesterday)
time(S,I4)

At S:

exists{I1,I2,I3}

C(I1,HumanBeings)
C(I2,Going)
C(I3,Cities)
agent(I2,I1)
destination(I2,I3)

(b) Equivalent Linear Notation.

Figure 3.6 Example of a Semantic Network Using a Paragraph.

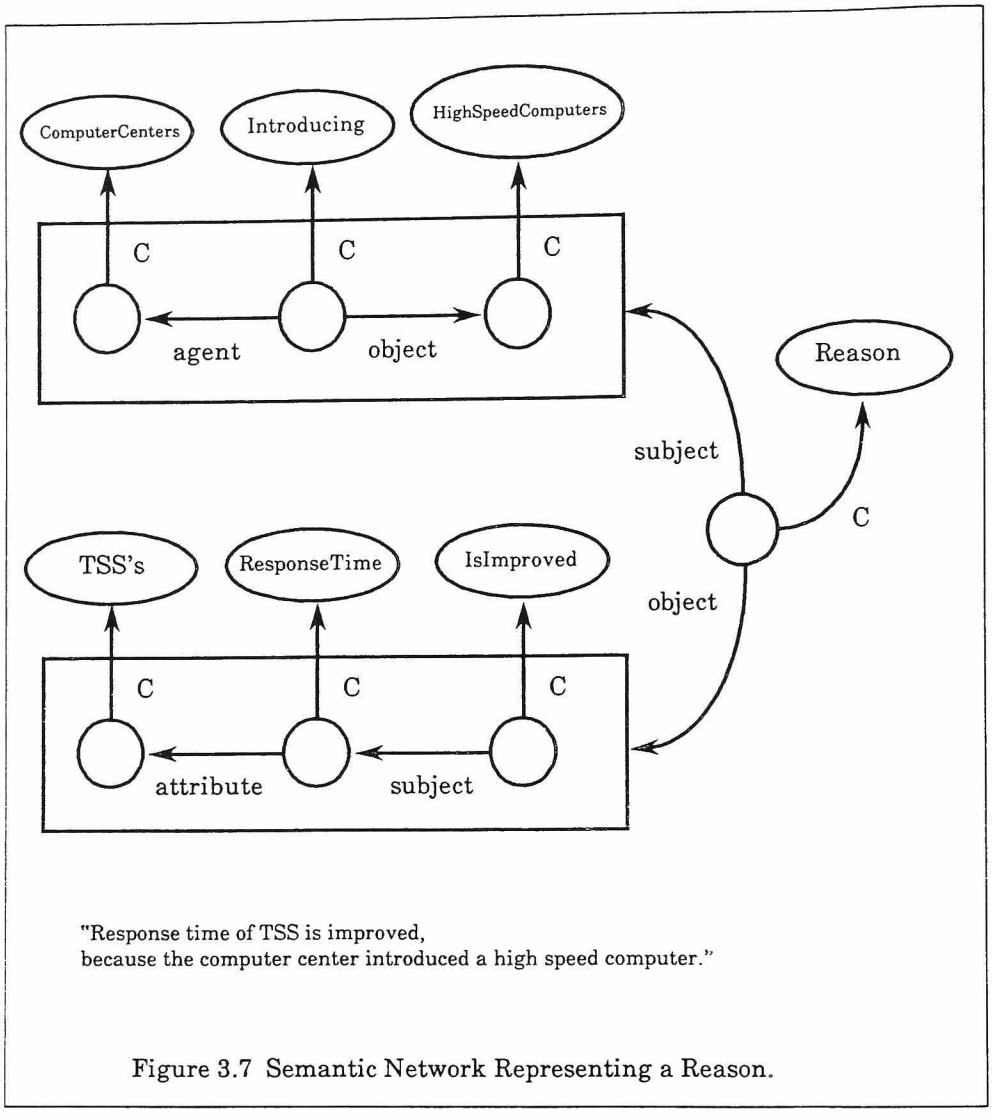
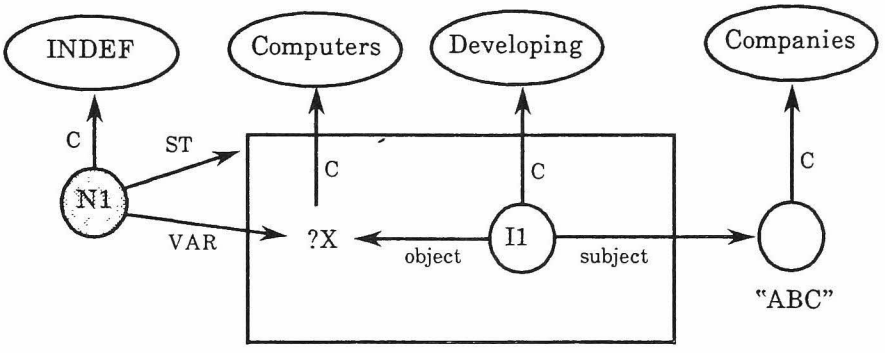


Figure 3.7 Semantic Network Representing a Reason.

Note that we consider that the *result* of the semantic interpretation is the same (shown in figure 3.9). The difference of the meaning as to which information is new and which information is presupposed is reflected by the difference of the process of creating semantic networks as to which piece of semantic network is newly created.

Transforming Intensional Structure into Extensional Structure



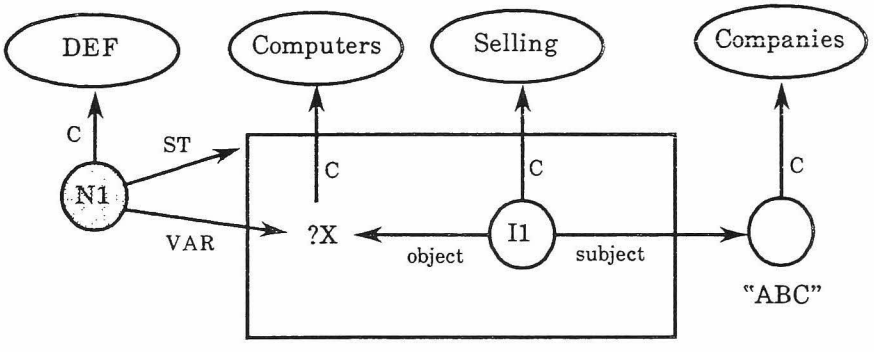
at the current context:

C(N1,INDEF)
ST(N1,S)
VAR(N1,?X)
...

at S:

exists {I1}

C(?X,Computers)
C(I1,Developing)
object(I1,?X)
subject(I1,ABC)



at the current context:

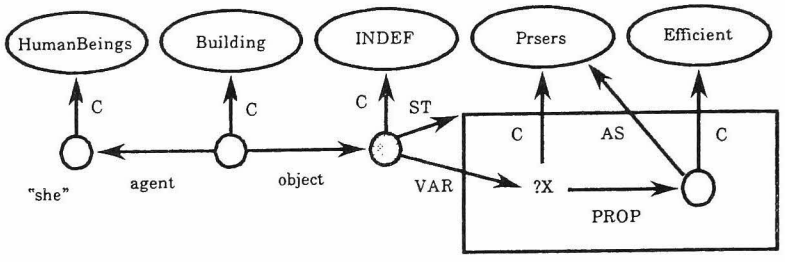
C(N1,DEF)
ST(N1,S)
VAR(N1,?X)
...

at S:

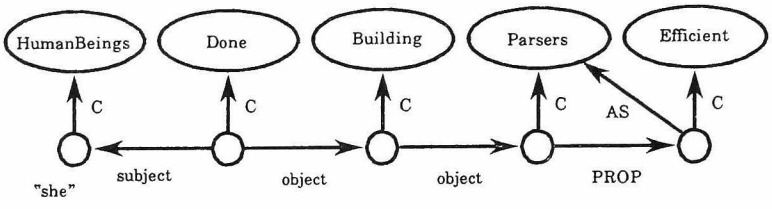
exists {I1}

C(?X,Computers)
C(I1,Selling)
object(I1,?X)
subject(I1,ABC)

Figure 3.8 Indefinite and Definite Representation.



(a) "She is building an efficient parser."



(b) "She has built an efficient parser."
(c) "She has built a parser which is efficient."
(d) "She has built a parser, which is efficient."
(e) "She has built a parser. It is efficient."

Figure 3.9 Semantic Networks for Connected Sentences.

As shown in the above examples, the existence of the described object depends on predication, such as aspect, tense, or use of specific verbs, like "succeed". Thus,

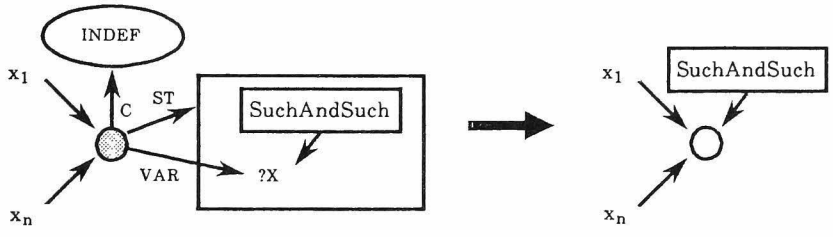
> She has succeeded in building a program which can understand natural language.

implies the existence of the program, and we can paraphrase it into two sentences:

> She has succeeded in building a program.
> The program can understand natural language.

In semantic network structures, we make the same sort of transformation, which we call *extensioning*. Figure 3.10 shows procedures for making extensioning for each intensional description we introduced.

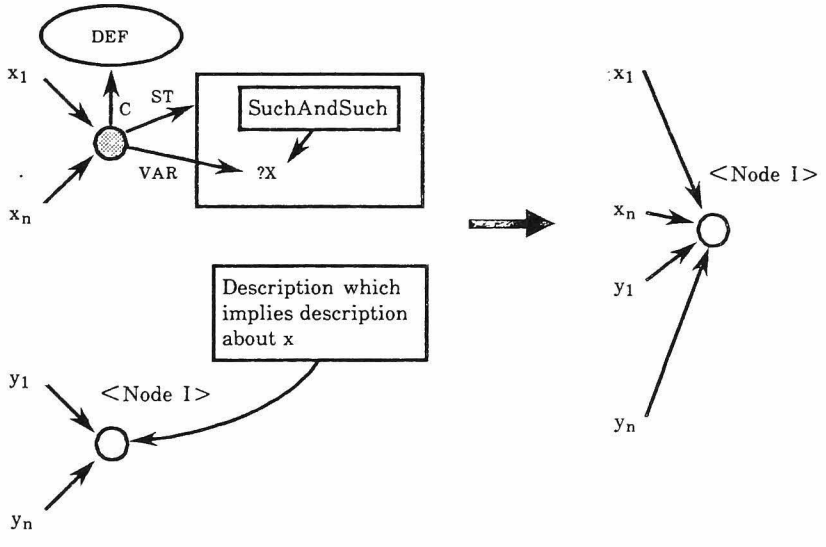
(a) Indefinite Node



(b) Definite Node:

Looking for an object which will match the specification.

If it is found, replace the definite structure by a node that is found.

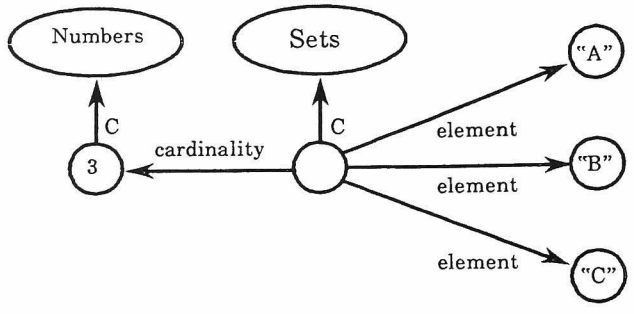


else ... same as the extensioning procedure for indefinite node (default processing).

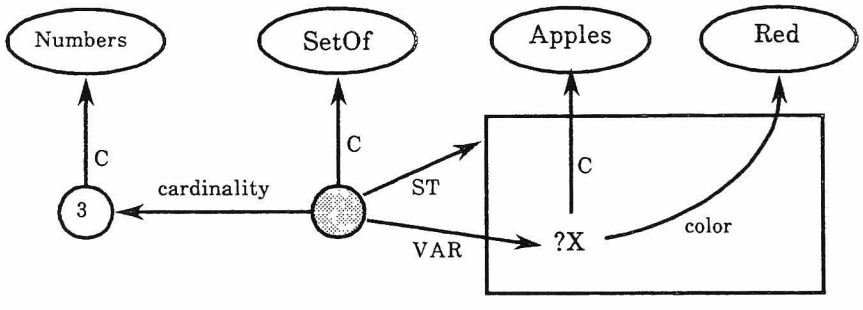
Figure 3.10 Extensioning Intensional Structure.

3.2.4.3 Set Nodes

In general, there are two ways for representing *sets*: extensional representation (e.g., {A, B, C}) and intensional representation (e.g., {x | P(x)}). Corresponding to this distinction, we represent sets in two ways. Figure 3.11a illustrates an example for extensional representaiton of *set*, while figure 3.11b for intensional representation of *set*.



(a) Extensional Representation: a set of A, B, and C.



(b) Intensional Representation: "three red apples".

Figure 3.11 Semantic Networks Representing *Sets*.

3.2.4.4 Quantification

Generally, there are two types of quantification: universal and existential. Existential quantification is implicit in our model; it is represented by the existence of nodes. On the other hand, we use special structure for universal quantification. Figure 3.12 illustrates some examples with equivalent logical formulas.

3.2.5 Procedure Attachment

In making semantic interpretations of natural language expressions, we have to refer to various kinds of knowledge to make inference. The nature of knowledge for interpreting natural language is considered to be lexicon-specific. Hence, we introduce to the knowledge base a lexical-driven feature by making a procedure attachment. Procedures can be attached to each class node. We sometimes classify attached procedures into *servants* and *demons* according to the notion introduced by Bobrow et al (1977). A servant type procedure will be invoked by a demand, while a demon-type procedure can be invoked in a data driven manner. Class nodes can share common knowledge using the convention of inheritence. From a class node, we can use procedures attached to its superclass nodes (figure 3.13).

3.3 Formal Representation for Natural Language

In this section we will describe the syntax and semantics of our formal representation EFR. Original notion of EFR is based on Cresswell's λ-categorial language[Cresswell 73]. Each expression of our formal language has a unique type, as introduced in previous sections.

3.3.1 Type

The set of types Syn is defined as a minimum set satisfying the following conditions:

(1) 0 and 1 $\in$ Syn, respectively.

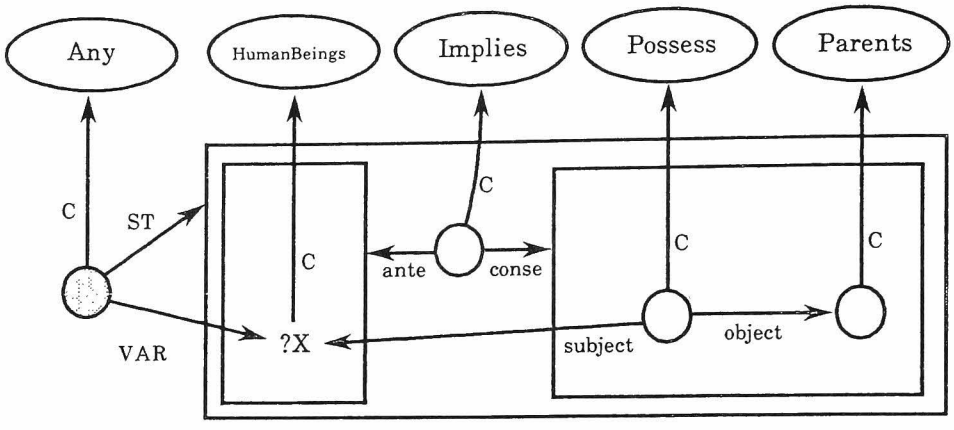
(2) if $\tau,\sigma_1, \ldots ,\sigma_n \in$ Syn then $<\tau,\sigma_1, \ldots ,\sigma_n> \in$ Syn, too.

According to this definition, set of types Syn is an infinite set as follows:

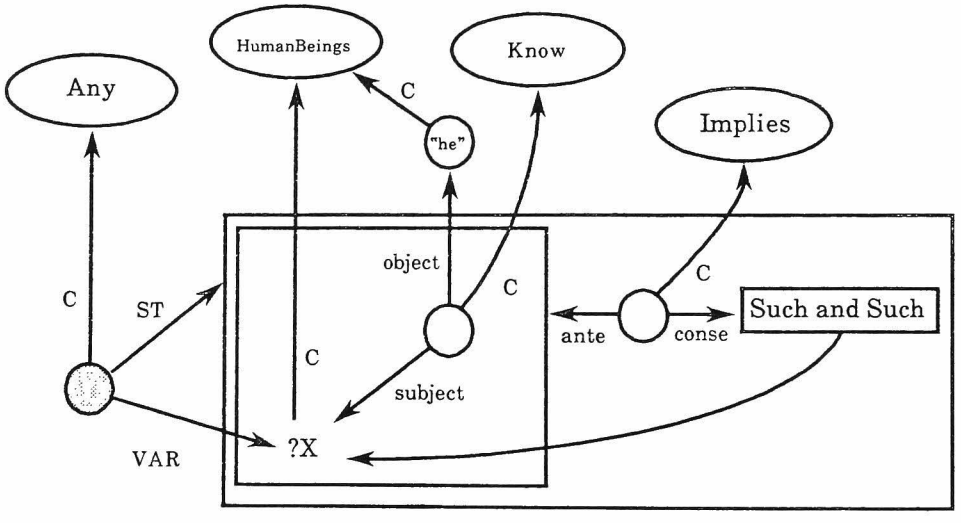
$$\text{Syn} = \{0, 1, <0,0>, <0,1>, <1,0>, <1,1>, \ldots, <0,<0,1>>, \ldots\}.$$

3.3.2 Syntax

Every expression in our formal representation is associated with exactly one type. Let E_α be the set of expressions of type α. Then our formal language L is



(a) ∀x[humanbeing(x) → ∃y[has(x,y) ∧ parent(y)]].



(b) ∀x[{humanbeing(x) ∧ know(x, "he")} → SuchAndSuch(x)].

Figure 3.12 Semantic Network Representations for Universal Quantification.

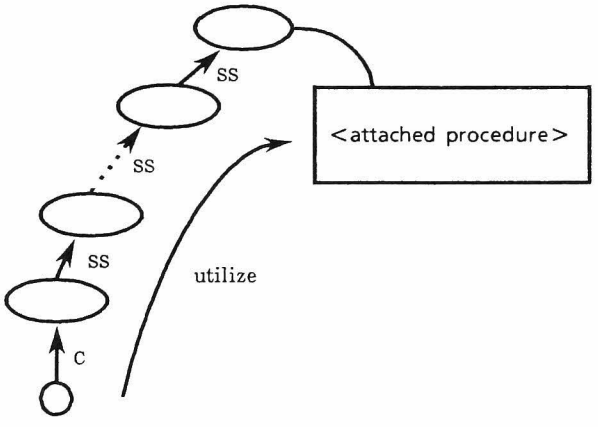


Figure 3.13 Inheritance of Attached Procedure.

defined as a union of E_α's. And the intersection of E_α and E_β must be empty if $\alpha \neq \beta$.

Expression of E_α is included in one of the following subsets:

constants of type α : F_α.

variables of type α : X_α.

composite expressions of type α.

The set of constants and variables is given *a priori*. It is assumed that an infinite number of variables are supplied. The set of well-formed composite expressions in the formal language are defined recursively using the following formation rules:

(rule 1) if δ is a well-formed expression which belongs to $E_{<\tau,\sigma_1, \ldots, \sigma_n>}$, and if $\alpha_1, \ldots, \alpha_n$ are well-formed expressions belonging to $E_{\sigma_1}, \ldots, E_{\sigma_n}$, respectively, then an expression $\delta(\alpha_1, \ldots, \alpha_n)$ is a well-formed expression in E_τ.

(rule 2) if β is a variable belonging to X_τ and if α is a well-formed expression in E_τ, then an expression $\lambda\beta[\alpha]$ is a well-formed expression in $E_{<\tau,\sigma>}$.

An example is shown below:

Let $E_{<0,1>}$: {ice-cream, run, walk, ... },
$E_{<0,1,1>}$: {have, eat, ... },
$E_{<0,<0,1>>}$: {she, he, ... },
$E_{<<0,1>,<0,1>>}$: {big, ... },
$E_{<<0,<0,1>>,<0,1>>}$: {a, the, ... },
X_1 : {x_1, y_1, ... }.

Then the below shows well-formed expressions derived from the above basic expressions:

(a) $big \in E_{<<0,1>,<0,1>>}$, ... by definition
(b) $ice\text{-}cream \in E_{<0,1>}$, ... by definition
(c) $big(ice\text{-}cream) \in E_{<0,1>}$, ... (a), (b), and (rule 1)
(d) $a \in E_{<<0,<0,1>><0,1>>}$, ... by definition
(e) $a(big(ice\text{-}cream)) \in E_{<0,<0,1>>}$, ... (c), (d), and (rule 1)
(f) $eat \in E_{<0,1,1>}$, ... by definition
(g) $x_1, y_1 \in X_1 \subset E_1$, ... by definition
(h) $eat(x_1,y_1) \in E_0$, ... (f), (g), and (rule 1)
(i) $\lambda y_1[eat(x_1,y_1)] \in E_{<0,1>}$, ... (g), (h), and (rule 2)
(j) $(a(big(ice\text{-}cream)))(\lambda y_1[eat(x_1,y_1)]) \in E_0$, ... (e), (i), and (rule 1)
(k) $\lambda x_1[(a(big(ice\text{-}cream)))(\lambda y_1[eat(x_1,y_1)])] \in E_{<0,1>}$, ... (g), (j), and (rule 2)
(l) $she \in E_{<0,<0,1>>}$, ... by definition
(m) $she(\lambda x_1[(a(big(ice\text{-}cream)))(\lambda y_1[eat(x_1,y_1)])]) \in E_0$ ... (k), (l), and (rule 1).

3.3.3 Semantic Interpretation

We defined semantic interpretation of our formal language as a process of building a semantic network structure which represents a new knowledge conveyed by a given expression in a given context.

Memory Space

Memory space for making semantic interpretation is classified into three categories according to their roles. STM or Short Term Memory is a memory space for storing information for an input sentence. The content of STM will be cleared when the interpretation of one sentence is completed. ITM or Intermediate Term Memory stores information about discourse. LTM or Long Term Memory stores

knowledge required for semantic interpretation as well as storing interpreted propositions. Figure 3.14 illustrates an example.

Functional Language for Manipulating Semantic Networks

Below are listed the constituents of our simple language for manipulating semantic networks. We use this language to describe operations on semantic networks.

Constants. A constant is an atomic symbol which begins with an upper case letter or or is placed in quotation marks. For example,

OBJECT, Object, or 'object'.

Variables. A variable is an atomic symbol beginning with a lower case letter. For example, the following items are variables:

x, y, z, ...

In particular, we use a variable "self" to represent self (instance) node and a variable "arg" to represent an argument handed to procedures.

Access path. Functional notation is used to represent an access path to other nodes. For example, we write,

adj(c(self))

to refer to a node reached by tracking down a link labeled "c" from the current node and going further along a link labeled "adj".

Operation to semantic network. There are a number of operations to create, modify, or evaluate a network structure. Major ones are:

Operations: instantiate[x], instantiate-variable and instantiate-paragraph instantiate a new instance node, variable, and paragraph in the current paragraph, respectively. A paragraph in which the node is placed may be explicitly indicated by an additional statement: "in z".

Operation: link[x,y,z] creates a new link from a node x to a node y with label z, in the current paragraph if it does not accompany additional statment such as "in s", which explicitly specifies where to create the link.

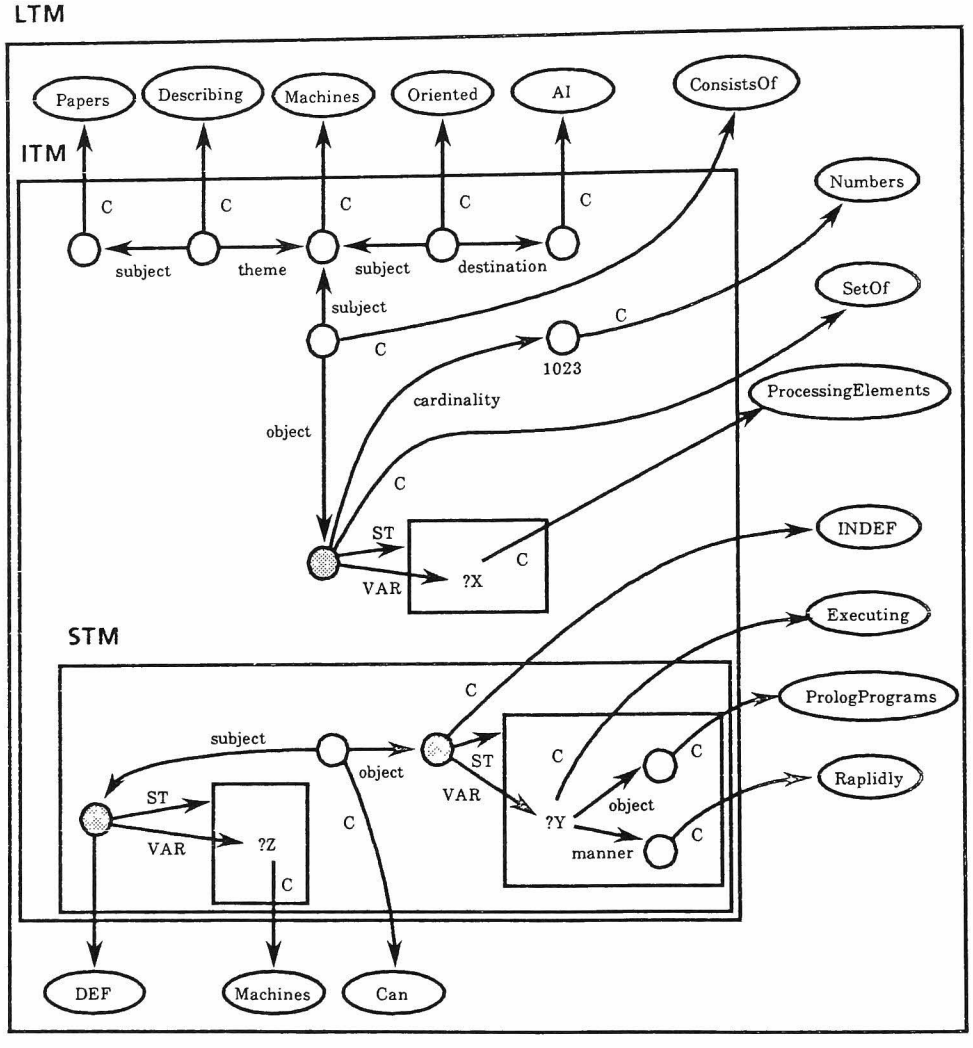


Figure 3.14 A Snapshot of a Memory Structure When the Third Sentence of the Following Text is Processed:

This paper describes a machine for AI.
The machine consists of 1023 processing elements.
The machine can execute PROLOG program rapidly.

Operation: evaluate[exp] evaluates the argument exp in the current paragraph.

- Operation: return[x] returns from the procedure with value x.

There are other operations, which will be explained when they appear.

Procedure for Semantic Interpretation

The procedure for interpreting expressions in our formal language consists of two steps:

(step 1) Replace each lexical item in a given expression by the appropriate formula of network manipulation language,

(step 2) Evaluate the resulting formula from outside to inside.

The above procedure is illustrated by example below. Let the expression be:

(a(dog))(walks)

Let the substitution be:

```
a ← λp[   x ← instantiate['INDEF"];
          y ← instantiate-variable;
          z ← instantiate-paragraph;
          link[x,y,VAR];
          link[x,z,ST];
          evaluate[p(y)] in z;
          return[x] ].

dog ← instantiate['dog_NOUN'].

walk ← instantiate['walk_VI'].
```

We make the following procedural attachment.

```
INDEF : to-apply:  evaluate[arg(self)] in the given context;
                   return[class-link-of[self]].

NOUN : to-apply: link[arg,class(self),C]; return[class-link-of[self]].
```

```
walking : to-apply: link[self,arg,AGENT];
                    extension[arg];
                    return[class-link-of[self]].
```

Figure 3.15 shows how the process of evaluation works.

3.4 Semantic Network Model for Natural Language

This section describes how we associate each syntactic structure of English with an expression in formal language and how we generate a semantic structure from those formal representations. EFR is a formal language which has a vocabulary of English.

We use the following notation to denote the correspondence between the phrase structure of English and expressions of EFR:

$$A \rightarrow B_1. \ldots .B_n \text{ where } <A> = f(<B_1>, \ldots , <B_n>).$$

In order to represent the deletion of English, we augment the above notation by introducing the following notation of a term:

(B-C)

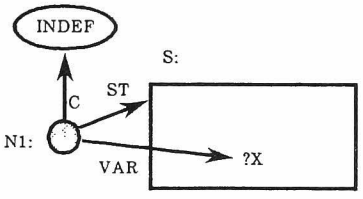
to denote a structure labeled B with exactly one structure labeled C taken out at some position. For example, we use a notation:

(S-NP)

to refer to the structures whose examples are shown in figure 3.16.

(step 1) Evaluating a(dog): The semantic network manipulation function for "a" is evaluated with p being instance node of class dog_{NOUN}.

(1) The first five steps will create the following network structure and variable binding:



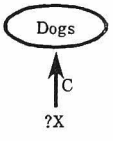
variable binding: x:N1, y:?X, z:S.

(2) The sixth step will evaluate the form p(y) in S, with

p: instance node of the class dog_{NOUN}

y: variable node ?X.

Then the procedure attached to the class node "NOUN" is invoked by means of inheritance, resulting in attaching a class link to the node bound to ?X:

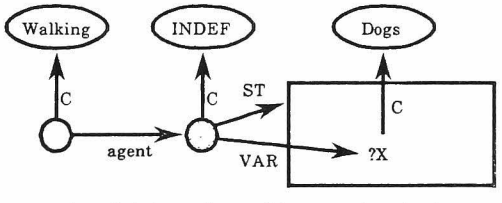


(3) The value is node N1.

(step 2) Evaluating (a(dog))(walks):

(1) A procedure attached to the class node INDEF is applied to an instance node of class "walking", resulting in the application of a procedure attached to the class node "walking" to the node N1.

(2) As a result, the following piece of semantic network is created:



(3) The above network is transformed by an extensioning operation, resulting in:

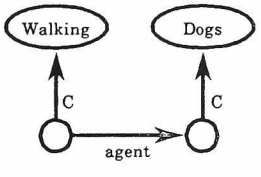


Figure 3.15 Generating Semantic Network as a Result of Semantic Interpretation.

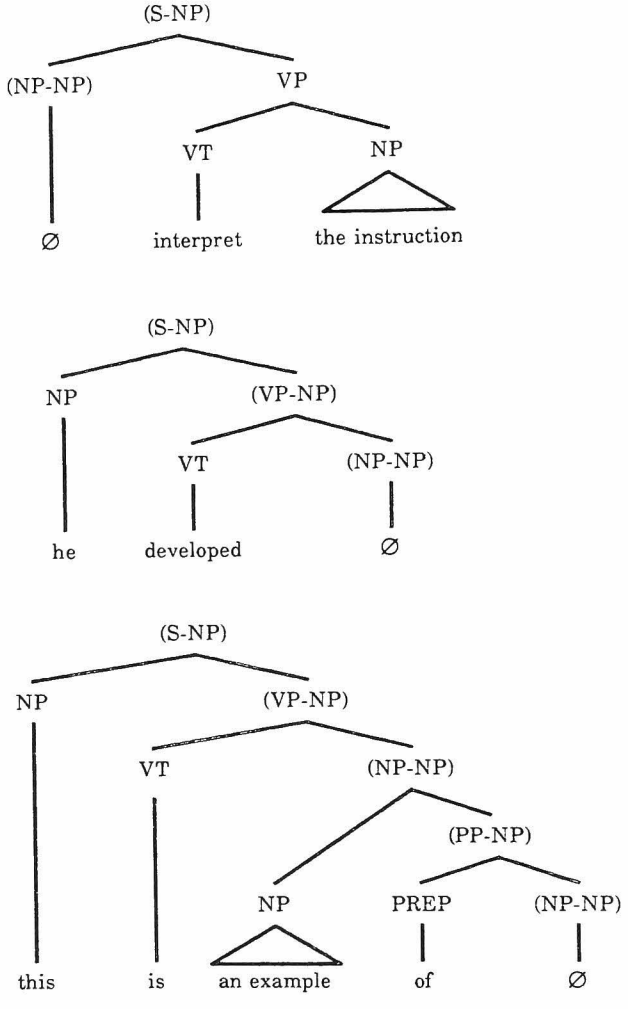


Figure 3.16 Example of Structures Denoted as (S-NP).

3.4.1 Simple Declarative Sentences

A simple sentence consists of a noun phrase and a verb phrase. Following Montague's analysis, we give a wider scope to the semantic representation for the noun phrase than that for the verb phrase:

$$S \rightarrow NP.VP \text{ where } <S> = <NP>(<VP>).$$

In this expression, <S> is of type 0, <NP> is of type $<0,<0,1>>$, and <VP> is of type $<0,1>$, respectively. In semantic interpretation, we think of a subject noun phrase as modifying the verb phrase by giving information about the subject case.

In Montague grammar, it is pointed out that this analysis can give an elegant solution to the local quantifier problem. Suppose our goal is to translate a sentence into first order logic. The problem is the treatment of quantification by determiners such as: "a", "the", "no", "every", "some", etc. From a syntactic point of view, those determiners are thought of as local constituents, while semantically their scope is global. For example, we want to make the following translation:

Everyone loves someone → $\forall x[human(x) \rightarrow \exists y[human(y) \wedge loves(x,y)]]$
or, $\exists y[human(y) \wedge \forall x[human(x) \rightarrow loves(x,y)]]$

The boy runs → $\exists x[boy(x) \wedge \forall y[boy(x) \leftrightarrow x=y] \wedge runs(x)]$

No student has a textbook → $\forall x[student(x) \rightarrow \neg \exists y[textbook(y) \wedge has(x,y)]]$.

Roughly speaking, Montague shows that this translation can be done within the constraint of compositionality: "the meaning of the whole is a function of the meaning of its parts". Montague's analysis is as follows:

Syntactic rule:

$S \rightarrow NP.VP$ where $<S> = <NP>(<VP>)$,
$NP \rightarrow DET.NOUN$ where $<NP> = <DET>(<NOUN>)$,
$VP \rightarrow VT.NP$ where $<VP> = \lambda x[<NP>(\lambda y[<VT>(x,y)])]$

Translation rule:

$a \leftarrow \lambda p[\lambda q[\exists x[p(x) \wedge q(x)]]]$
$the \leftarrow \lambda p[\lambda q[\exists x[p(x) \wedge \forall y[p(x) \leftrightarrow x=y] \wedge q(x)]]]$
$no \leftarrow \lambda p[\lambda q[\forall x[p(x) \rightarrow \neg q(x)]]]$

every ← λp[λq[∀x[p(x) → q(x)]]]
everyone ← λq[∀x[human(x) → q(x)]]
someone ← λq[∃x[human(x) ∧ q(x)]]
student, textbook, runs, has, loves ←
    student, textbook, runs, has, loves, respectively.

Using these rules, logical expressions can be obtained mechanically from the input sentence. Note that we could obtain the same result using a rule:

$$S \rightarrow NP.VP \text{ where } <S> = <VP>(<NP>).$$

In this analysis, <VP> is thought of as denoting a function adding information to the denotation of <NP>. The difference seems to be only in formalism and the degree of complication, and not in any analytical power.

3.4.2 Verb Phrases

The type of expression associated with a verb phrase is <0,1>, a type denoting one place predicates. The basic structure of a verb phrase consists of a main verb and a number of subsequent noun phrases:

VP → VI where <VP> = <VI>
VP → VT.NP where <VP> = λx[<NP>(λy[<VT>(x,y)])]

A semantic network manipulation function assigned to a verb phrase as a whole will generate a semantic network for a proposition from a given individual node referred to by a noun phrase of the subject. We assign to a main verb an n-place predicate. For example,

```
have ← λxy[  z ← instantiate['have'];
             link[z,x,SUBJECT];
             extension[x];
             link[z,y,OBJECT];
             extension[y];
             return[current-paragraph] ].
```

Note that, in the above example, subject case value and object case value will be replaced by extensional structures, for we can presuppose their existence.

3.4.3 Simple Noun Phrases

A noun phrase consists of a determiner and a noun:

NP → DET.NOUN where <NP> = <DET>(<NOUN>).

Interpreting Nouns

In interpreting a noun, we first associate it with an instance node of a class whose superclass is a class NOUN. For example,

$$\text{computer} \leftarrow \text{instantiate}[\text{'computer}_{\text{NOUN}}\text{'}].$$

If this node is applied to an instance node of class individual, a link will be created to indicate the class of the individual node. We attach a procedure to a class node NOUN to do this operation (figure 3.17).

What is implied in this operation is that if $p_{NOUN}(x)$ is asserted for an individual x, then we build a semantic network asserting that x belongs to a subclass p of individuals. The link labeled "den" from a subclass of NOUN to a subclass of individuals represents this relationship. Thus we have a dual relationship in noun concepts and individual concepts:

$$\forall x[p_{NOUN}(x) \rightarrow q_{NOUN}(x)] \Leftrightarrow p \subseteq q.$$

Interpreting Determiners

Figure 3.18 shows semantic network manipulation functions we assign to determiners. Notice that those assignments will generate the intensional structures we introduced in earlier section.

Interpreting Possessives

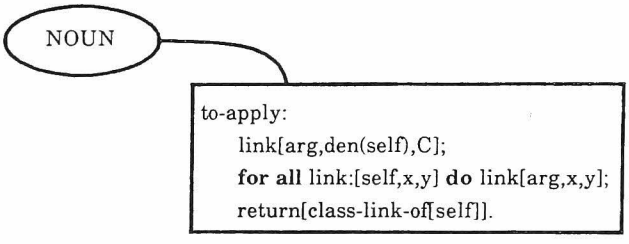
The internal structure of possessives consists of a noun phrase and a possessive morpheme " 's ". As the function of a possessive is considered to be the same as that of a determiner, we make the following analysis:

$$\text{DET} \rightarrow \text{NP.'s where } <\text{DET}> = \lambda n[\lambda p[<\text{NP}>(\lambda x[((*\text{poss}(x))(n))(p)]]].$$

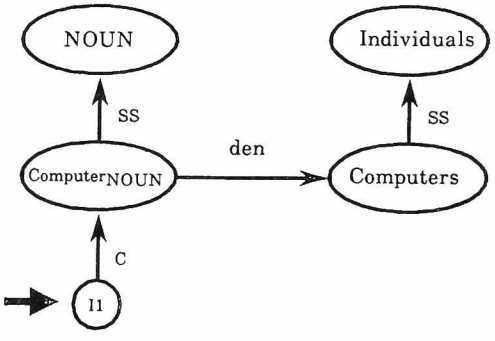
We interpret this EFR expression as follows:

"the *n* which possesses *p*".

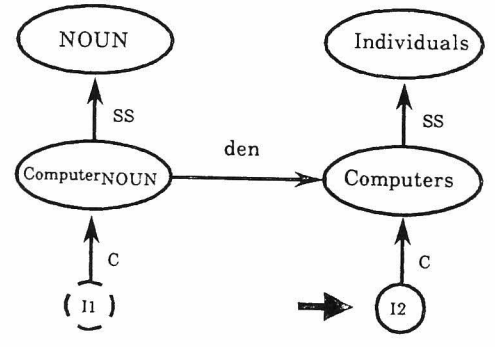
In order to do this, we introduce a class node "*poss" and "possessive" with procedures attached to them (figure 3.19).



(a) Procedure Attachment to a Class Node "NOUN"



(b) Structure of Semantic Network Resulting from Interpreting a Noun "computer", (node I1).



(c) Structure of Semantic Network Resulting after the Node I1 is applied to an Individual Node I2.

Figure 3.17 Procedure Attachment for a Class NOUN and Process of Semantic Interpretation of Nouns.

```
a ← λp[    x ← instantiate['INDEF'];
           y ← instantiate-variable;
           z ← instantiate-paragraph;
           link[x,y,VAR];
           link[x,z,ST];
           evaluate[p(y)] in z;
           return[x] ].

the ← λp[      x ← instantiate['DEF'];
               y ← instantiate-variable;
               z ← instantiate-paragraph;
               link[x,y,VAR];
               link[x,z,ST];
               evaluate[p(y)] in z;
               return[x] ].

every ← λp[λq[   x ← instantiate['ANY'];
                 y ← instantiate-variable;
                 z ← instantiate-paragraph;
                 a ← instantiate-paragraph;
                 c ← instantiate-paragraph;
                 link[x,y,VAR];
                 link[x,z,ST];
                 i ← instantiate[IMPLIES] in z;
                 link[i,a,ANTE] in z;
                 link[i,c,CONSE] in z;
                 evaluate[p(y)] in a;
                 evaluate[q(y)] in c;
                 return[class-link-of[x]] ]].
```

Figure 3.18 Assignments of Semantic Network Generation Function to Determiners.

Using those nodes we make an assignment:

*poss ← instantiate['*poss'].

Figure 3.20 shows an example as to how a possessive is interpreted.

Actually, we attach an additional procedure to deal with the cases in which an abstract head noun is used. The problem is to recover the semantic relationship

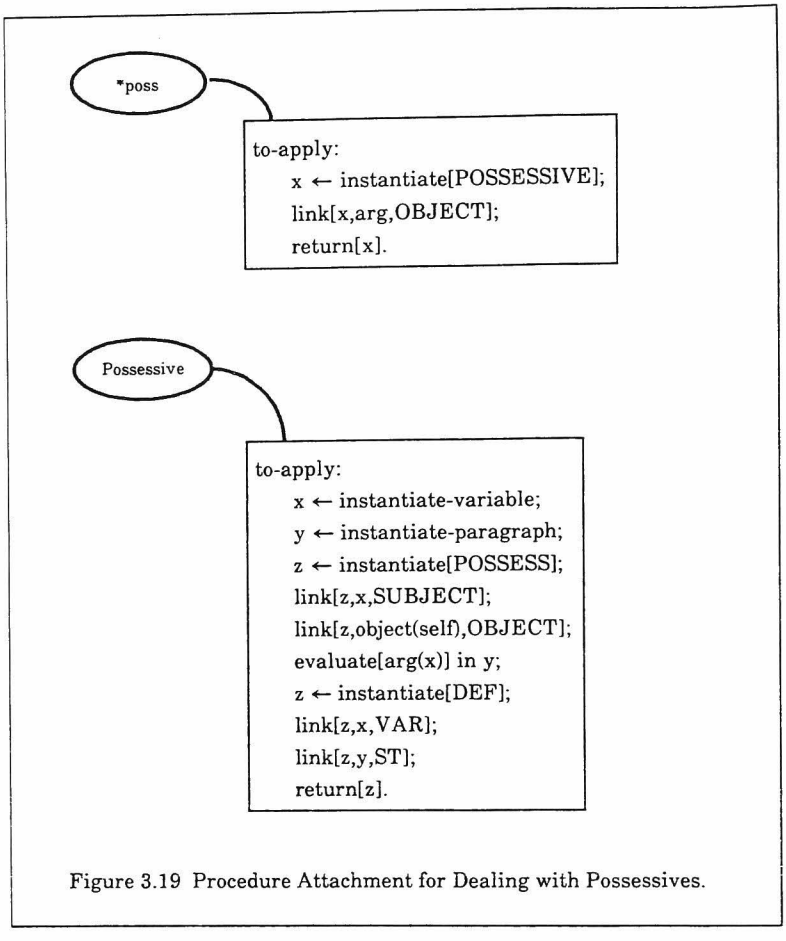


Figure 3.19 Procedure Attachment for Dealing with Possessives.

between the noun phrase in the possessive and the modified noun. For example, from a phrase:

"his collaboration"

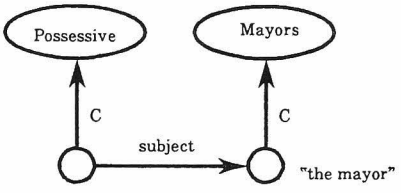
we have to recognize that "he" is the subject of an activity "collaboration". In order to make this inference, we attach a procedure to a class node "possess". Figure 3.21 shows semantic network transformation by this attached procedure.

3.4.4 Noun Modifiers

There are various noun modifiers in English: adjectives, adjectival prepositional phrases, relative clauses, present and past participles. Although these modifiers can be regarded as a modifying phrase to the noun phrase in terms

input: "the mayor's house"
EFR: (the(mayor))(λx[((*poss(x))(house))(p)])

Evaluation Process:
(1) Result of evaluating *poss(x):



(2) Result of evaluating (*poss(x))(house):

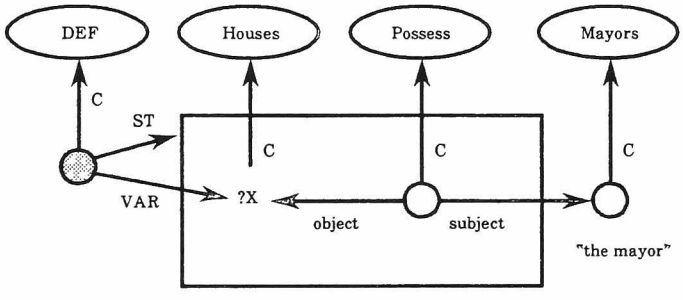


Figure 3.20 Example of Interpreting an EFR Expression for Possessives.

of constituent analysis, we analyze them as a modifier to the head noun, for this analysis is more adequate from the semantic point of view. Thus we write:

NOUN → ADJ.NOUN where <NOUN> = <ADJ>(<NOUN>)
NOUN → NOUN.PP_{ADJ} where <NOUN> = <PP_{ADJ}>(<NOUN>)
NOUN → NOUN.RELCL where <NOUN> = <RELCL>(<NOUN>)
NOUN → NOUN.VP_{ING} where <NOUN> = <VP_{ING}>(<NOUN>)
NOUN → NOUN.VP_{EN} where <NOUN> = <VP_{EN}>(<NOUN>).

This section elaborates the internal structure of these modifying phrases of nouns.

Adjectives

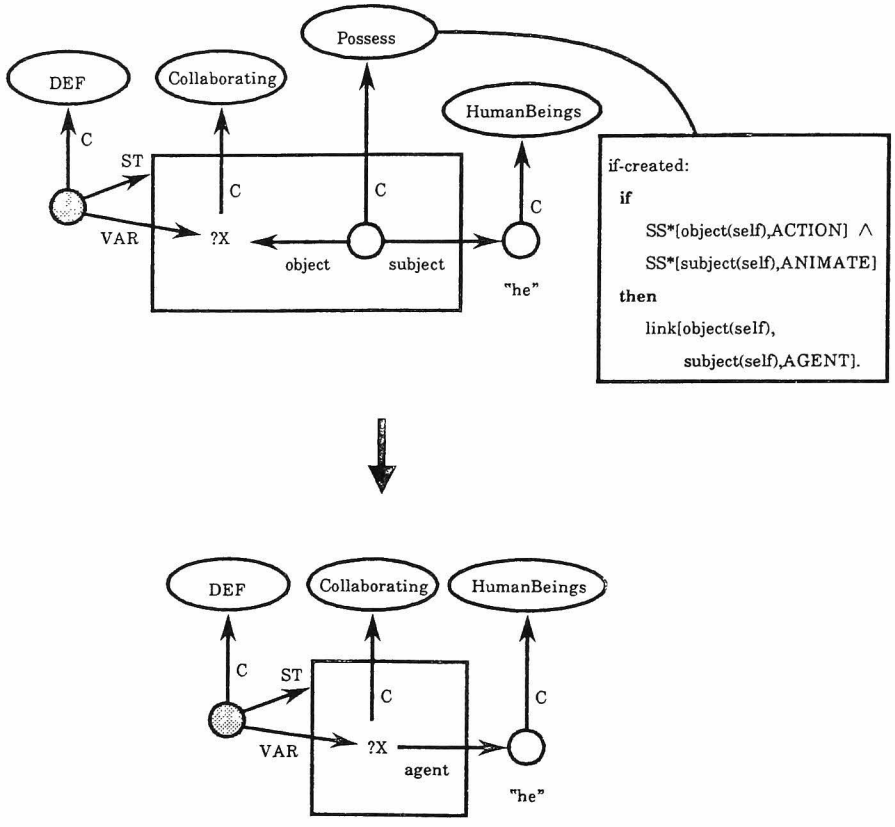


Figure 3.21 An Attached Procedure and Semantic Network Transformation for Possessives.

As to interpretation of adjectives, several problems have been pointed out:

(a) Adjectives cannot be treated as a one-place predicate: for example,

"big ant" vs "small elephant".

(b) Problems of idiomatic expression: for example,

"hot dog", "thunder bird", "fire bird", etc.

(c) Treatment of intensional adjectives: such as:

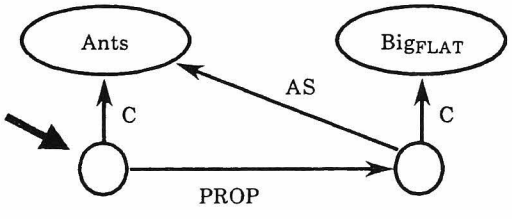
"former president", "fake gun", etc.

Here, we consider the first two problems.

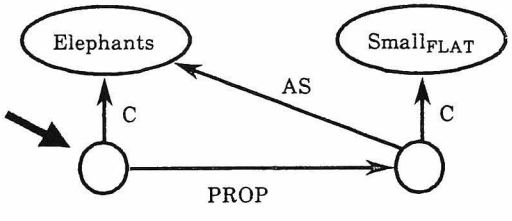
To deal with the first problem, we take an adjective as a two-place predicate. For example,

$$\text{a big animal} \Leftrightarrow \exists x[\text{animal}(x) \wedge \text{big}_{FLAT}(x, \text{'animal'})].$$

We specify that the predicate "$\text{big}_{FLAT}(x,y)$" is true iff an individual denoted by x is "big" *as* y. In semantic networks, we use the structure shown in figure 3.22.



(a) Semantic Network for "a big ant"



(b) Semantic Network for "a small elephant".

Figure 3.22 Samantic Network for Adjectives.

Here we introduced a couple of new links: AS and PROP links. A relation represented by an AS link from X to Y:

as(X,Y)

indicates that a property denoted by X depends on Y. On the other hand a relation represented by a PROP link from X to Y:

prop(X,Y)

indicates that X has a property Y.

In order to build the semantic network structure, we make an assignment like:

big ← instantiate['big'],

and attach the following procedure to the class ADJECTIVE:

```
to-apply: x ← instantiate[flat(c(self))];
          link[arg,x,PROP];
          link[x,c(arg),AS];
          return[arg].
```

Note that each adjective node can use this procedure by means of inheritance.

Using the structure assigned to adjectives, we can answer the question:

"how big is it?",

without any ambiguity. This is true even if afterwards an additional class link is attached to the individual node, as a result of interpreting a sentence like:

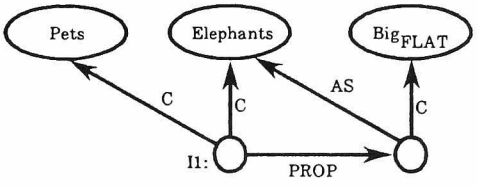
"it is a pet".

Figures 3.23 (a) and (b) show the structure and a procedure to extract the answer, respectively.

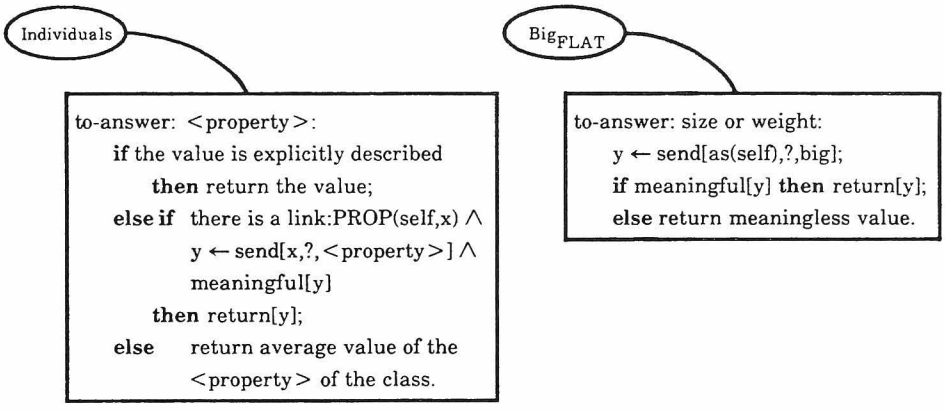
If we used a structure as in figure 3.23(c), it might be ambiguous as to whether the node I is as big as a pet (say 100Kg in weight), or as big as an elephant (say 1000Kg in weight).

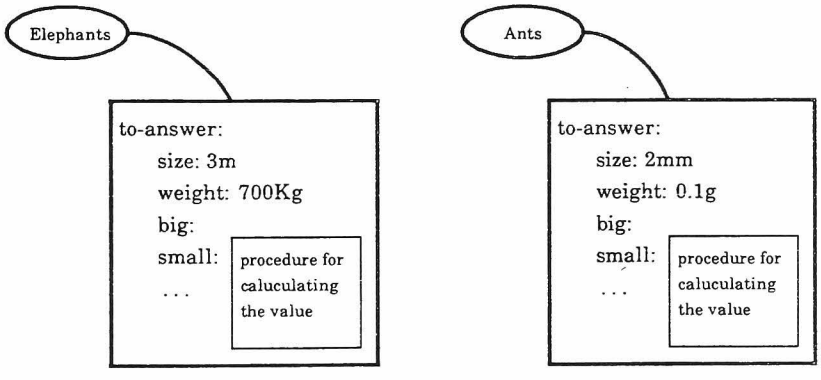
As to the second problems, there are two possible solutions. One is to think of those expressions simply as idiomatic expressions and to merge them into one word in the morphological analysis or the syntactic analysis phase. This treatment may be reasonable in practice. The other possibility is to deal with this problem in the semantic interpretation phase. This can be done in our framework using a conditional form in an attached procedure (figure 3.24).

But note that this treatment is essentially the same as those done in syntactic analysis.

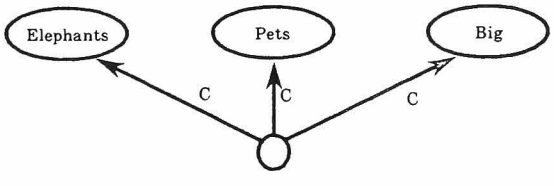


(a) "I1 is a big elephant, and it is a pet."





(b) Procedure Attachment for Retrieving Information.



(c) Structure which involves Ambiguity.

Figure 3.23 Semantic Netowrk and Procedure Attachment for Adjectives.

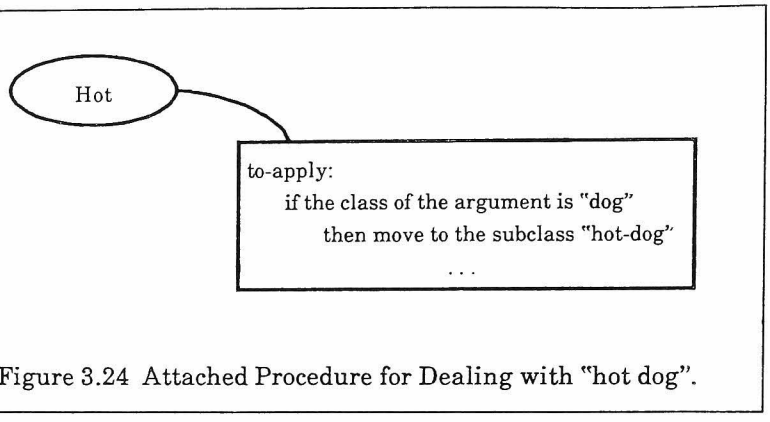


Figure 3.24 Attached Procedure for Dealing with "hot dog".

Adjectival Prepositional Phrases

Adjectival prepositional phrases as a whole are analyzed as modifiers of nouns. Their constituent structure consists of a preposition followed by a noun phrase.

$PP_{ADJ} \rightarrow$ PREP.NP where $<PP_{ADJ}> = \lambda n[\lambda x[<NP>(\lambda y(((*ap(<PREP>))(y))(n))(x)])]]$

In semantic interpretation, we try to simplify the fairly complicated expression for prepositional phrases. The basic idea is as follows:

(a) interpret <PREP> as an operator which takes an individual and which will create a node functioning as an adverb,

(b) interpret *ap as an operator which takes an instance node introduced above and which will create another operator functioning as an adjective generator,

(c) accordingly, a subexpression *ap(<PREP>) is interpreted as an operator which takes an individual and which will create a node functioning as an adjective,

(d) subexpression (*ap(<PREP>))(y) is interpreted as a compound adjective, and subsequent processing is the same as that for adjectives.

Figure 3.25 illustrates this processing for an example sentence.

The sort of inference we want to make in interpreting adjectival prepositional phrases is to recognize the semantic relationship between the head noun and the object of the prepositional phrase. For example,

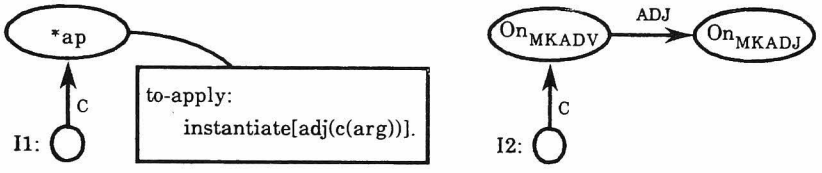
input: "(a) house on the hill"

EFR: λx[(the(hill))(λy[((((*ap(on))(y))(house))(x)])]

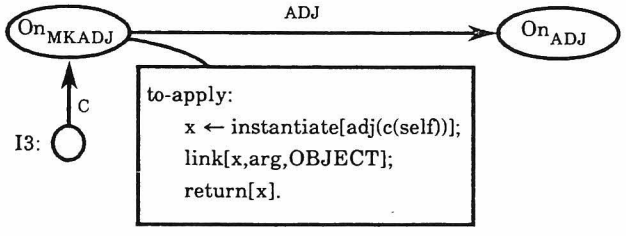
substitutions:

```
    . . .
*ap ← instantiate['*ap'];
on ← instantiate['on_MKADV'];
    . . .
```

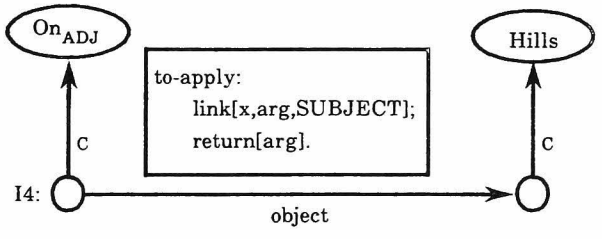
evaluation process:



(1) Corresponding to *ap(on), node I1 is applied to I2, resulting in the node I3 below.



(2) Corresponding to (*ap(on))(y), I3 is applied to I4, an instance node referred to by "the(hill)", resulting in:



(3) Corresponding to ((*ap(on))(y))(house), I4 is applied to node I5, the result of interpreting a noun "house".

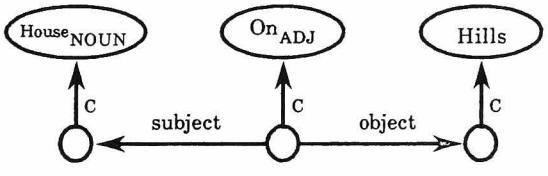


Figure 3.25 Semantic Interpretation of Adjectival Prepositional Phrase.

(1) "the analysis of natural language" → "natural language" specifies the object case of "analysis".

(2) "the analysis by a computer" → "a computer" specifies the agent case.

(3) "the weight of the baby" → "the baby" specifies the object of the weight attribute.

Figue 3.26 shows procedure attachment for making these inferences and how these problems are solved.

Relative Clauses

The constituent structure of a relative clause consists of a relative pronoun (RELPRON) and a clause with exactly one noun phrase deleted (denoted as (S-NP)).

RELCL →RELPRON.(S-NP) where <RELCL> = which(<(S-NP>)

The expression of EFR for (S-NP) is of type <0,1> or the type denoting a one-place predicate. The operator "which" maps type <0,1> expressions into type <<0,1>,<0,1>> expressions (type denoting adjectives).

In order to give semantic interpretation to the above form, we make the following substitution:

```
which ← λp[λq[λi[   evaluate[p(i)] in the current context;
                    evaluate[q(i)] in the current context;
                    return[current-context] ]]].
```

Figure 3.27 illustrates an example.

In the case of a relative adjective "whose", we make a similar treatment. The association rule is:

RELCL → whose.NOUN.(S-NP) where <RECCL> = (whose(<NOUN>))(<S-NP>).

The assignment is:

```
whose← λn[λp[λq[λi[   x ← instantiate[individual];
                      y ← instantiate[POSSESS];
                      link[y,x,OBJECT];
                      link[y,i,SUBJECT];
```

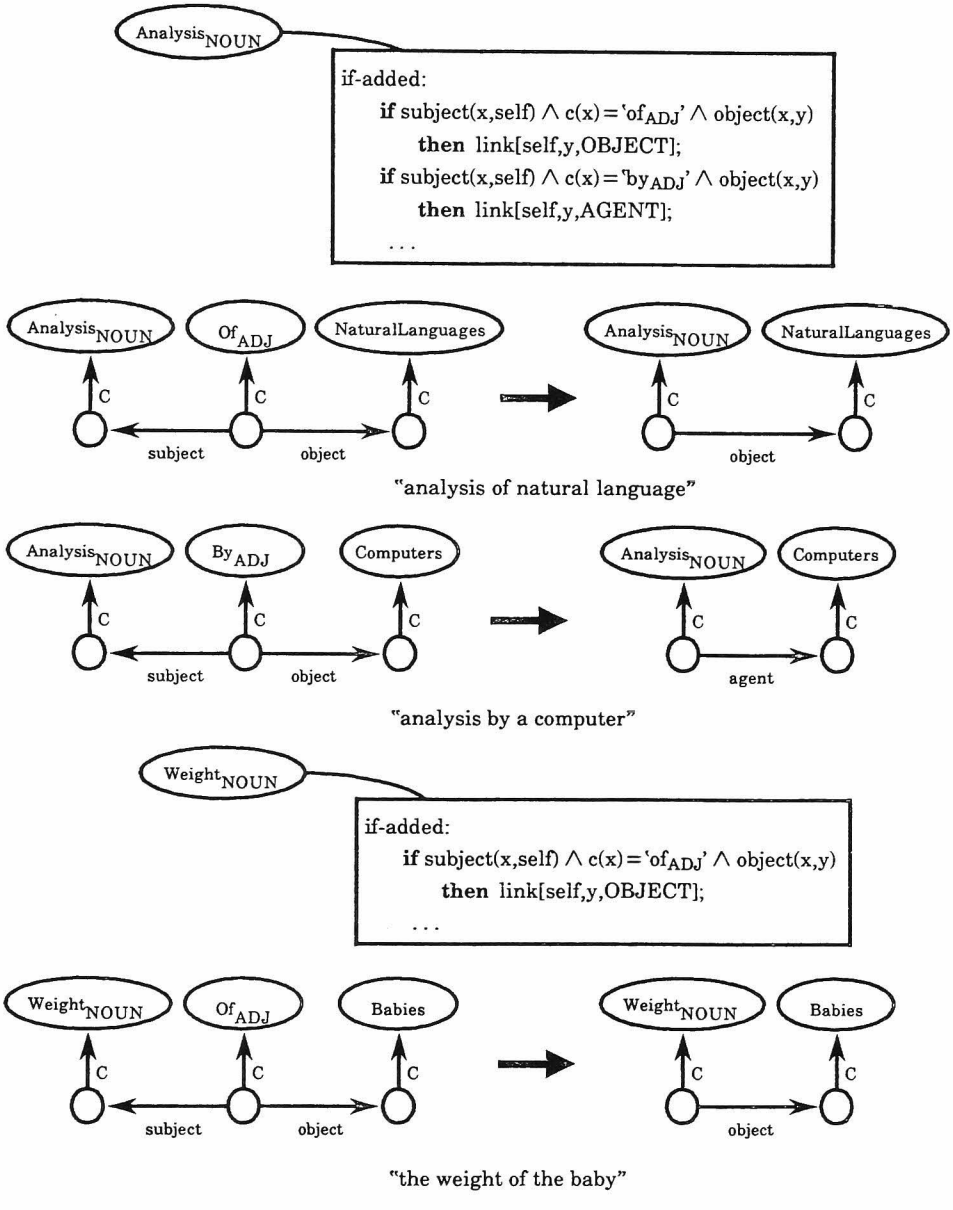


Figure 3.26 Procedure Attachments for Making Inference about the Semantic Role of Adjectival Prepositional Phrases.

evaluate[q(i)] in the current context;
evaluate[p(i)] in the current context;

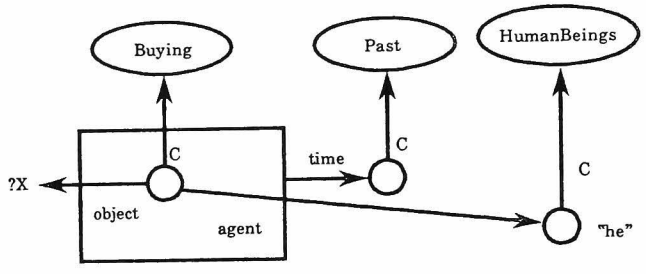
input: "the car which he bought"

EFR: the((which(λy[he(did(λx[buy(x,y)])]))(car))

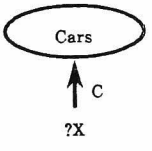
Evaluation:

(1) Evaluation of "the" generates a variable node, say ?X.

(2) Subexpression: λy[he(did(λx[buy(x,y)]))] evaluates to a one place predicate which takes an individual node and which will generate a semantic network for proposition. In this example, a variable node ?X is given as an argument and the following structure is generated:



(3) "car" also evaluates to a one-place predicate, which is in turn applied to the variable node ?X. This attaches to ?X a class link pointing to the class "car"



(4) As a result, the following semantic network is generated:

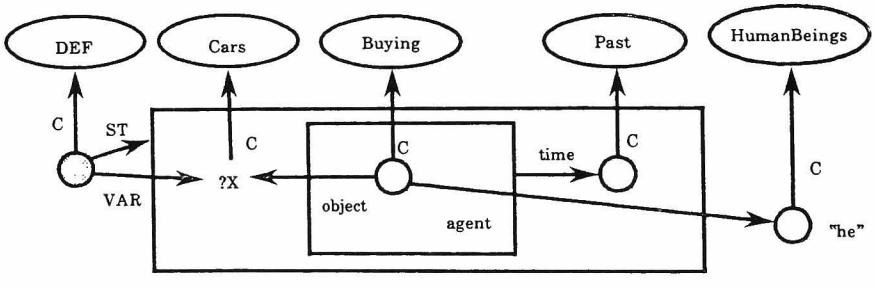


Figure 3.27 Example of Interpretation of a Relative Clause.

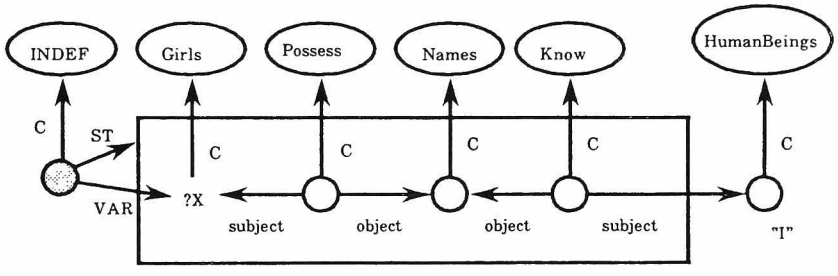
evaluate[q(i)] in the current context;
return[current-context].

Figure 3.28 shows an example.

input: "a girl whose name I know"

EFR: a(((whose(name))(λy[I(λx[know(x,y)])])))

Semantic Network Generated:



Note that this semantic network will be further simplified by a procedure attached to the class node "possess".

Figure 3.28 Example of Interpreting "whose".

Present and Past Particles

Semantic representation for present and past particles looks like:

$$<VP_{ING/EN}> = parti(<VP>).$$

A lexical item "parti" is an operator of type $<<<0,1>,<0,1>>,<0,1>>$, denoting a function from one place predicates to adjectives.

In semantic interpretation, we assign the same semantic network manipulating function as that assigned to a relativizer "which". This corresponds to the following approximation:

"a girl singing a song" → "a girl who sings a song,"

"the house broken by the storm" → "the house which was broken by the storm."

3.4.5 Sentence and Verb Phrase Modifiers

Adverbs

In our formal language, sentence adverbs and verb phrase adverbs are not distinguished by type. We simply take both as basic expressions of type $<0,0>$. Thus,

$S \rightarrow ADV.S$ where $<S> = <ADV>(<S>)$,
$VP \rightarrow ADV.VP$ where $<VP> = \lambda x[<ADV>(<VP>(x))]$,
$VP \rightarrow VP.ADV$ where $<VP> = \lambda x[<ADV>(<VP>(x))]$.

In semantic interpretation, we make a distinction. Figure 3.29 shows an example.

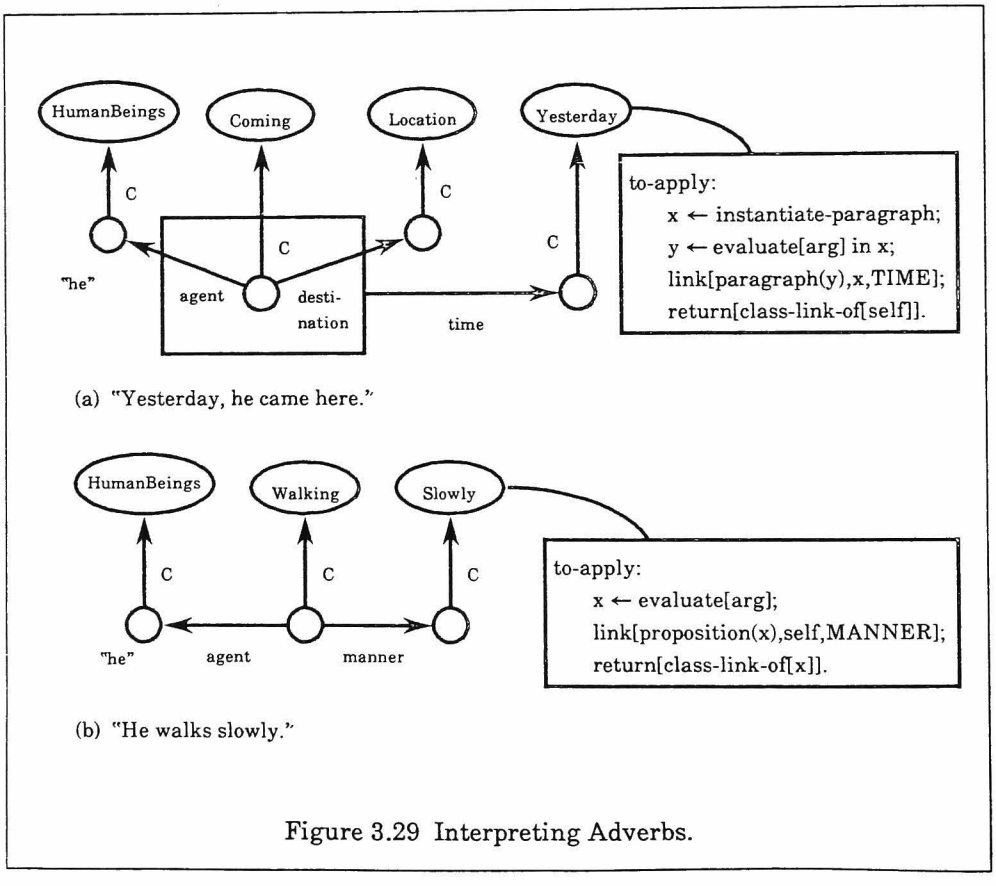


Figure 3.29 Interpreting Adverbs.

Adverbial Prepositional Phrases

Prepositional phrases can function as adverbials. The EFR expression we associate with a prepositional phrase as an adverbial differs a bit from that for a prepositional phrase as an adjective:

$PP_{ADV} \rightarrow$ PREP.NP where $<PP_{ADV}> = \lambda n[\lambda x[<NP>(\lambda y((<PREP>(y))(n))(x))]]]$.

The interpretation of the connection between the preposition and an object noun phrase is almost the same as for adjectival prepositional phrases. In particular, the assignment to <PREP> is the same. Figure 3.30 shows an example.

Subordinate Clauses

Syntactically, a subordinate clause consists of a subordinate conjunctive and a clause. Subordinate clause as a whole are treated as modifiers of a sentence, giving information about condition, time, reason, etc. The association rule for subordinate clauses is:

S → SCL.S where <S> = <SCL>(<S>),
S → S.SCL where <S> = <SCL>(<S>),
SCL → SCONJ.S where <SCL> = <SCONJ>(<SCL>).

In semantic interpretation, a network structure connecting a couple of paragraphs is generated. We illustrate an example in figure 3.31.

3.4.6 Noun Clauses

Noun clauses consist of a complimentizer and an embedded sentence:

NP → THAT.S where <NP> = that(<S>).

An operator "that" is of type $<<0,<0,1>>,0>$.

Semantic Interpretation of a noun clause will generate a paragraph structure. Figure 3.32 illustrates an example.

Other structures of noun clauses are possible, as indirect questions. The treatment of indirect questions will be described later as related to interrogatives.

3.4.7 Mood

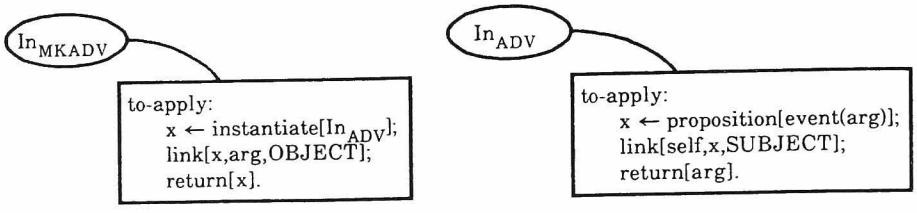
Input: "He bought it in London."

EFR: he(did(λx[(in(London))(it(λy[buy(x,y)])))]))

Assignment:

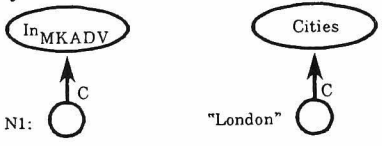
```
. . .
in ← instantiate[In_MKADV].
. . .
```

Procedure Attachments:

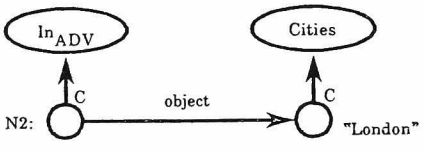


Process of Evaluation:

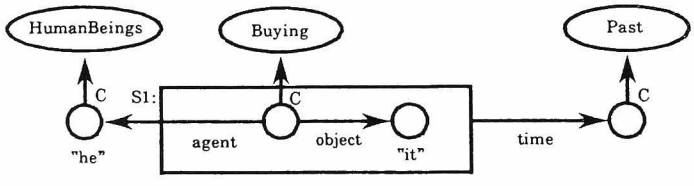
(1) Evaluating "in" and "London" result in the instance nodes N1, and "London", respectively.



(2) Node N1 is applied to the node "London". Procedure attached to the node In_{MKADV} is invoked. As a result a new node N2 is created.



(3) Embedded proposition is evaluated and a paragraph S1 is created.



(4) Node N2 is applied to the paragraph S1. Procedure attached to the node In_{ADV} is invoked and finally, we get the following network structure:

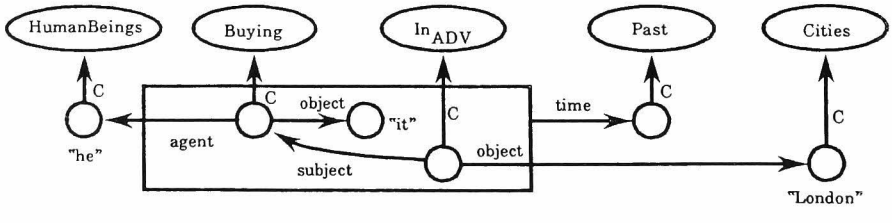


Figure 3.30 Semantic Interpretation of Adverbial Prepositional Phrase.

input: "As a typhoon is coming, he repairs his house."

EFR: (as((a(typhoon))(ing(come))))(he(λx[(his(house))(λy[repair(x,y)])])))

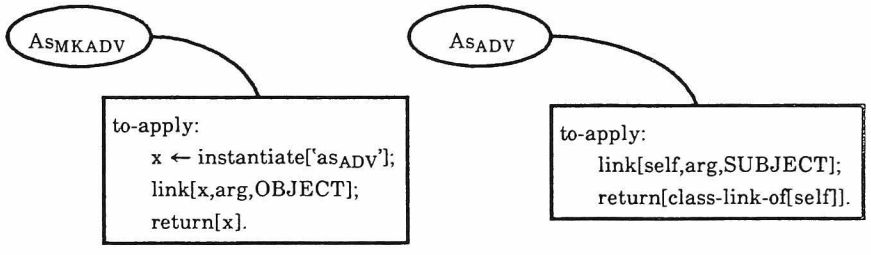
Substitutions:

. . .

as ← instantiate['as$_{MKADV}$'].

. . .

Procedure Attachments:



Result of Evaluation:

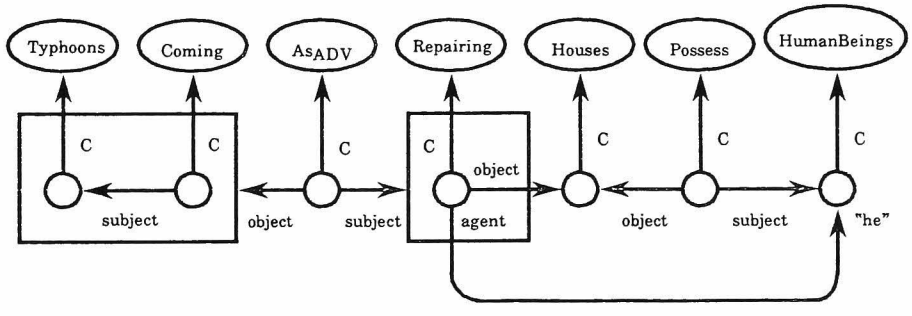


Figure 3.31 Interpreting Subordinate Clauses.

3.4.7.1 Interrogatives

There are various constructions of interrogative sentences. Here we consider only typical cases.

Yes/No Questions

An yes/no question is initiated by an auxiliary verb:

input: "He believes that she is a mayor."

EFR: he(λx[that(she(λy[(a(mayor))(λz[is(y,z)])]))])

Substitutions:

. . .

that ← [x ← instantiate-paragraph;
evaluate[arg] in x;
return[x].]

. . .

Procedure Attachment:

paragraph: | to-apply:
evaluate[arg(self)] in the current context. |

Result of Evaluation:

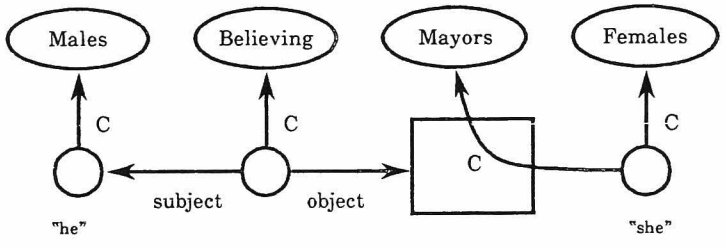


Figure 3.32 Interpreting Noun Clause.

S → AUX.NP.VP where <S> = #ques(whether(<NP>(<VP>))).

The semantic network we will generate from an yes/no question corresponds to the following statement:

"the speaker asks the hearer about the truth value of the proposition denoted by <NP>(<VP>)."

(example)

input sentence: "did he attend the project?"

EFR: #ques(whether(he(did(λx[the(project))(λy[attend(x,y)])]))))

Semantic network is shown in figure 3.33.

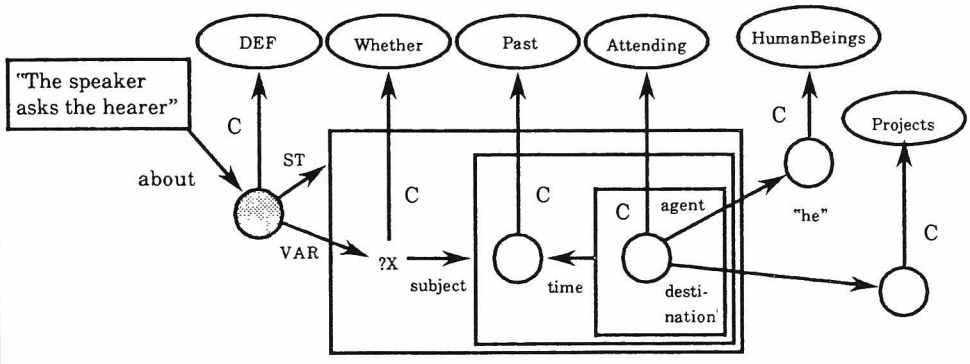


Figure 3.33 A Semantic Network for a Question:
"Did he attend the project?"

WH-questions (1)

Here we deal with the case in which an interrogative is initiated by words like "when", "where", or "why". The association rule is:

S → QADV.AUX.NP.VP where <S> = #ques(<QADV>(<NP>(<VP>))),

vocabulary words of category <QADV> involve "when", "where", "why".

We make the following paraphrase:

"the speaker asks the hearer about the attribute value of f(<QADV>) of the proposition,"

where, f(when) = time, f(where) = place, f(why) = reason.

(example)

input sentence: "where did he get it?"

EFR: #ques(where(he(did(λx[it(λy[get(x,y)])])))))

The semantic network for this sentence is shown in figure 3.34.

WH-questions (2)

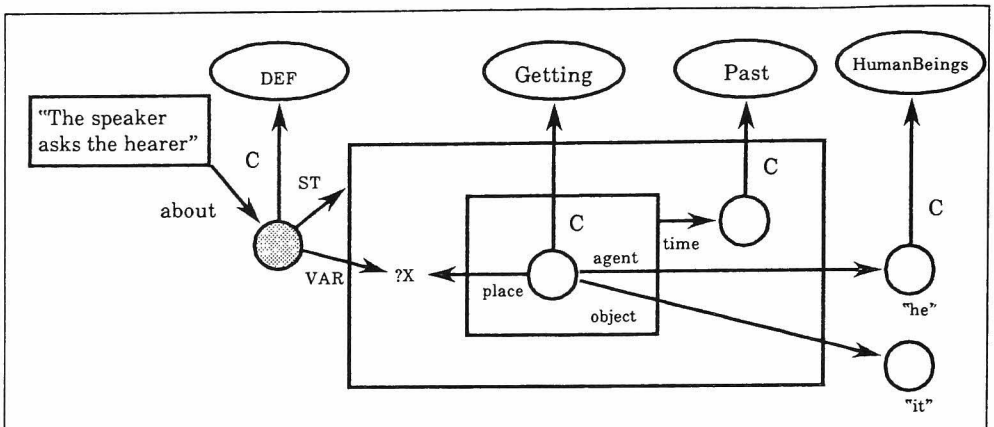


Figure 3.34 A Semantic Network for a Question:
"Where did he get it?"

We deal with the case in which the question is initiated by phrases like, "what", "which <NOUN>", or "how <ADJ> <NOUN>". The association rule is:

$S \rightarrow QNP.AUX.NP_1.(VP\text{-}NP_1)$
where $<S> = \#ques(<QNP>(\lambda y[<NP_1>(\lambda x[(<VP\text{-}NP_1>(y))(x)])]))$,
$S \rightarrow QNP.VP$ where $<S> = <QNP>(<VP>)$,

vocabulary words of category QNP involve "what".

$QNP \rightarrow QDET.NOUN$ where $<QNP> = <QDET>(<NOUN>)$,

vocabulary words of category QDET involve "which",

$QDET \rightarrow how.ADJ$ where $<QDET> = how(<ADJ>)$.

The paraphrase we make for those structures is roughly as follows:

"the speaker asks the hearer about the denotation of <QNP> which has a property denoted by the type <0,1> expression following it."

(example 1)

input sentence: "What does the company supply?"

EFR: $\#ques(what(\lambda y[(the(company))(\lambda x[((\lambda v(\lambda u[supply(u,v)]))(y))(x)])]))$
$\equiv \#ques(what(\lambda y[(the(company))(\lambda x[supply(x,y)])]))$

The semantic network generated from this expression is shown in figure 3.35.

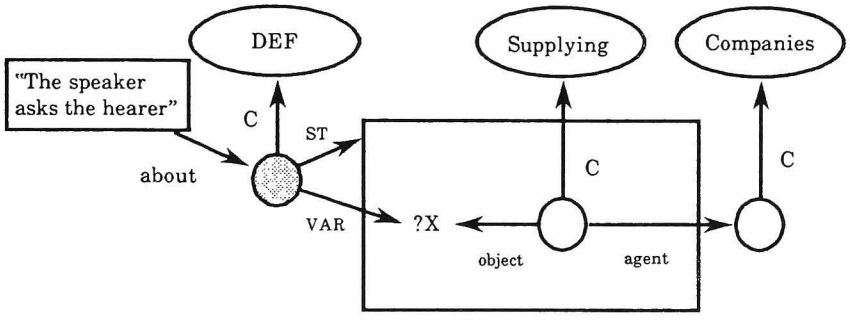


Figure 3.35 Semantic Network for a Question:
"What does the company supply?"

(example 2)

input sentence: "Which company supplies the project?"

EFR: (which(company))(λx[(the(project))(λy[supply(x,y)])])

The semantic network generated from this expression is shown in figure 3.36.

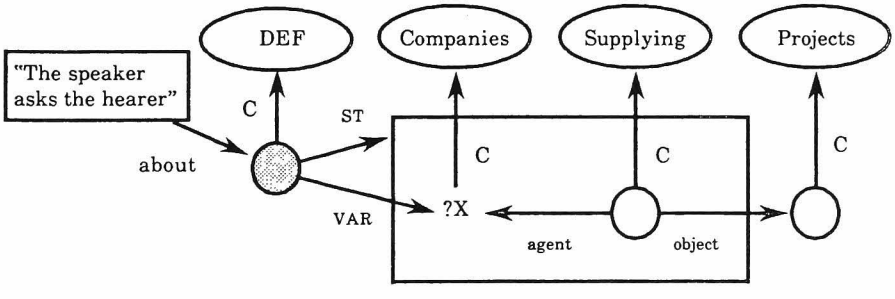


Figure 3.36 Semantic Network for a Question:
"Which company supplies the project?"

(example 3)

input sentence: "How many products does the shop deal with?"

EFR: ((how(many))(*pl(product)))(λy[(the(shop))(λx[deal-with(x,y)])])

The semantic network generated from this expression is shown in figure 3.37.

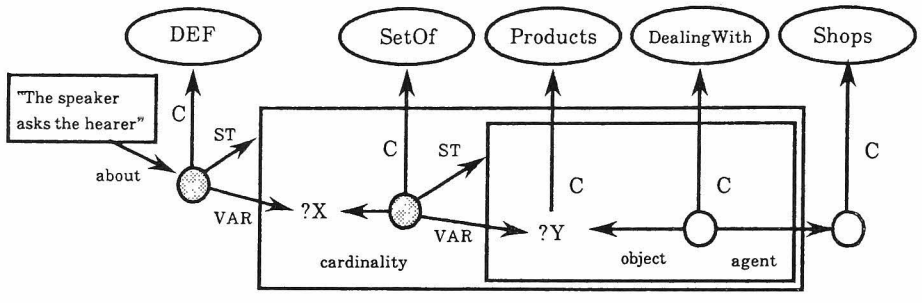


Figure 3.37 Semantic Network for a Question:
"How many products does the shop deal with?"

3.4.7.2 Imperatives

Imperatives consist of a verb phrase:

S → VP where <S> = #imp(inf-n(<VP>)).

As in interpreting interrogatives, we paraphrase them into a declarative sentence:

"the speaker orders the hearer to make an action denoted by <VP>."

(Example)

input sentence: "Take off your coat"

EFR: #imp(inf-n(λx[(your(coat))(λy[take-off(x,y)])]))

The semantic network generated from this expression is shown in figure 3.38.

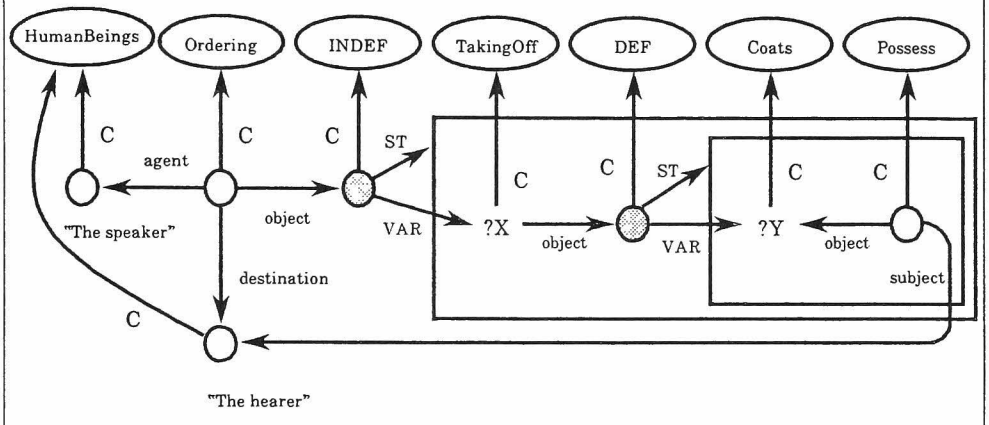


Figure 3.38 Semantic Network for an Imperative:
"Take off your coat!"

4. Translating English into Formal Representations

4.1 Introduction to Chapter 4

This chapter discusses the issues in translating English expressions into expressions of EFR. The main topic is concerned with parsing. We elaborate the architecture of a parser.

4.2 Basic Design of the Parser

Machine translation system requires a large grammar to analyze input sentences. It also requires knowledge about concepts, pragmatics, or task domain in order to resolve ambiguities. At the current state of the art, however, no comprehensive grammar nor knowledge base is available. What we can do is to invent an interaction mechanism to supplement incomplete knowledge of the parser as well as to expand grammar and the knowledge base by trial and error. The following design goals are considered to be important in making these activities efficient:

(a) grammar rules must be easy to write,

(b) modularity of grammar rules,

(c) efficiency in time and memory space required,

(d) effective man-machine interaction at parsing time,

(e) utilities for dictionary management.

4.2.1 Issues in Designing a Parser

There are several issues to be addressed in designing a parser.

Declarative Model vs Procedural Model

Let a symbol L stand for a language to be analyzed. Conventional parsers for natural language analysis can be classified into two major categories.

Declarative Model: Parser = G(L) + Parsing Engine, where G(L) stands for a grammar for L. In a parser of this category, a grammar for generating well formed sentences of L is given in terms of a phrase structure grammar. Thus a language L itself is specified declaratively. Given a sentence, the parsing engine looks for a series of rules in G(L), which is considered to derive the given sentence. Usually, efficient parsing algorithms for context free language are utilized. In most parsers, phrase structure grammar is somehow extended to deal with phenomena specific to natural languages. Examples of this category involve Extended-LINGOL[Tanaka 77], DCGS[Pereira 80], DIAMOND[Robinson 80], and LSP[Sager 81].

Procedural Model: Parser = A(L) + Rule Interpreter, where A(L) stands for an automaton which accepts sentences in L. Parsers of this category use rule languages to describe instructions to the automaton. A(L) is identified with a description in the rule language. Thus a language itself is implicitly specified as a set of sentences which can be accepted by the automaton A(L). A parser of this category analyzes input sentence by executing the instructions specified in A(L). This category can be further classified into ATNG or augmented transition network grammars[Woods 70] and situation-action parsers[Winograd 83]. ATNG is the most popular formalism in artificial intelligence research. Word Expert Parser[Rieger 79] and PARSIFAL[Marcus 80] contains the latter aspect.

There are tradeoffs of simplicity of rule writing and flexibility of parsing. With the declarative model, we can write a grammar more easily, for we have only to give a description about how the phrase structures of the language look like. Also, modularity is attained, for augmented phrase structure rules are almost independent of each other. One can modify, delete, or add a rule without affecting most of the other rules.

This should be contrasted with rule writing with procedural rules, in which we have to specify *what to do* in each situation. Writing a procedural grammar compels the rule designer to make more mental effort than writing a declarative grammar. Additionally, a procedural rule may interact with other rules. So we have to be careful enough to make the rule set take the desired behavior in each situation.

On the contrary, a procedural model can provide a more efficient and flexible strategy for controlling the parsing process. Certain parsers are discussed in relation to a performance model of natural language recognition by human beings. The procedural model is sufficient for handling phenomena such as: idiomatic expressions, verb patterns, and various types of agreements, which cannot be suitably handled by uniform application of declarative rules.

Top Down vs Bottom Up

Parsing algorithms are classified into top down or goal oriented versus bottom up or data driven ones. Most current natural language parsers combine these two factors, so the essential difference between the two seems to disappear.

Backtrack vs Parallel

A chronological backtracking algorithm is said to be inefficient, for it may do the same processing repeatedly. On the other hand, the problem with a parallel algorithm is the explosion of memory space required to hold every intermediate result. This aspect becomes problematic when the constraint of the analysis grammar is weak and potentially lots of ambiguities exist for each input sentence. Unfortunately, such situation must be unavoidable in the early stages of a grammar development. Some systems utilize a score driven best first search, but this approach is dangerous in terms of completeness. With a chronological backtracking algorithm and a parallel algorithm, the parser is complete in the sense that all possibilities will be tried once and only once. With other heuristic search algorithms, the designer or rule writer has to elaborate a mechanism to assure the completeness of the grammar, if he or she wants safety.

4.2.2 Basic Design Strategies

Initial Rule Writing in Declarative Rules and Parsing with Procedural Rules

Our parser uses situation-action rules at the parsing time. Initially, however, the rule writer can describe the structures of the language to be analyzed as a set of augmented phrase structure rules. There are simple principles for transforming those augmented phrase structure rules into situation-action rules. Initial versions of situation-action rules transformed from augmented phrase structure rules can be used directly for parsing, but they can be modified, augmented, or improved by incorporating additional control structure.

Combining Top Down and Bottom Up Strategy

The bottom up feature is incorporated both to deal with idiomatic expressions or other data driven features and to avoid endless expansion of left recursive goals in top down parsing.

The top down feature is incorporated to deal with inherited features, which are transmitted from parent nodes to descendant nodes.

Backtracking

The backtracking facility is used to save memory space even if the grammar produces a lot of ambiguous results. To improve efficiency, the parser saves all intermediate results and utilizes them if the same goals are encountered again. This aspect is common to parsing strategies of chart parsers.

Interactive Diagnosis

The parser analyzes an input sentence and tries to produce possible analyses one by one. The user can simply reject the analysis when it is not acceptable, and he can order the parser to seek other possibilities. Additionally, the user can tell the parser which part of the result is wrong and which part is right. The parser uses this information to obtain the desired analysis faster than otherwise. We call this facility an interactive diagnosis.

Utilities for Dictionary Management

In designing a large grammar, we might change the information format as well as the content of certain classes of dictionary entries. To assist this activity, we install a pattern directed dictionary management utility.

4.3 Describing the Syntax and Semantics of a Language

We use augmented phrase structure grammar to represent a declarative description of syntax and semantics of the language to be analyzed. Each node of the phrase structure tree is augmented by a list of attribute-value pairs (feature complex). Special notations are introduced to augment the expressive power of phrase structure grammar. We use following type of rule description:

R_n: $A \rightarrow \beta_1 \ldots \beta_n$ where <feature operations>.

This is a natural augmentation of the syntax-semantics association rules introduced in chapter 3. Now semantic representation is associated with a specific

feature: SEM. Other features are used for constraining the possible combination of phrases.

Use of Features to Constrain Possible Combinations of Phrases

Using the feature complex, we can rule out improper analysis by checking illegal combination of phrases. Actually, we use only the so-called syntactic features shown in table 4.1.

The set of current features should be extended to include semantic features. But unlike syntactic features, semantic features seem to be useful only for making preference and not for rejecting certain combinations of phrases. For example, the agent case of a verb "drink" seems to be restricted to animate objects. But we can say,

"our table will order dinner"

meaning that

"the guests at our table will order dinner."

Dealing with such problems involves many difficulties and it does not seem to be beyond the scope of this dissertation.

Representing Holes

In describing English, we need to have a syntactic device for "holes". Some grammar formalism uses the notion of meta rule to give a notation to holes[Gazder 83]. Others heavily use features to make it possible to give precise characterization to holes[Winograd 72]. We extend context free grammar so as to give straightforward description to the phenomenon of holes. In the extended framework, we can write a rule like:

$$NP \rightarrow NP_1.\text{which}.(S\text{-}NP_1)$$

for relative clauses. The extended notation $(S\text{-}NP_1)$ denotes a structure S with only one NP deleted. The subscript is used to identify the deleted structure. Thus it is indicated that the deleted NP has the same syntactic and semantic property as the head NP. In a structure governed by $(S\text{-}NP_1)$, a dummy NP node is used as a place holder. Although it is a null word, it is given the same feature complex as the head NP. Accordingly, we can rule out phrases such as:

Table 4.1 Syntactic Features and their Meaning.

Attribute	PossibleValue	Meaning
COMP	{+ER,+EST,NIL}	indicates the inflectional suffix of adjective.
CONJ	{AND,OR,NOR,NIL}	quotes a conjunctive.
COUNT	{ABLE, UNABLE}	indicates countability of noun.
CTYPE	{CO, SUB}	represents the type of a conjunctive, CO: Coordinating conjunctive, SUB: Subordinating conjunctive.
FORM	{+ED, +EN, +S, +ING, ORG, AM, ARE, IS, WAS, WERE}	stands for verb form.
FORM2	{WAS, WERE, NIL}	represents the tense of a verb.
G-CASE	{+'S, NIL}	indicates whether a noun is -s genitive or not.
NBR	{SGL, PL}	represents number of a noun: SGL: singular, PL: plural.
NBR2	{(SGL), (PL), (SGL PL)}	represents range of a noun's number compatible with a determiner.
NBRNP1	{T, NIL}	indicates whether a noun is modified by a numeral or not.
NOT	{NOT, NIL}	indicates whether "not" is attached or not.
NR	{T, NIL}	indicates whether a clause is nonrestrictive or not.
NTYPE	{INF, NOM, THAT, NIL}	indicates the type of a noun phrase and noun clause.
PMOD	{A,B, C, D, NIL}	indicates subcategorization of a verb.
PREP	<preposition>	quotes preposition.
PSN	{1,2,3}	indicates person.
SUBJ	<EFR expression for subject>	this attribute value is used for filling an omitted subject.
UIAPROP	{T, NIL}	indicates whether a pronoun can be used as a postmodification.
VP	<EFR expression for a verb phrase>	this attribute is used for supplementing an omitted verb phrase.

*those boys who has a bicycle.

A more complicated example is illustrated below:

NP → NP_1.$PREP_1$.which.(S-(PP ($PREP_1$ NP_1))).

Since this rule checks the agreement of the prepositions, we can accept:

the language into which this sentence is translated,

for the same reason as a sentence:

this sentence is translated into the language.

is acceptable. But a sentence:

the language with which this sentence is translated,

can not be accepted using the same reasoning (figure 4.1). Thus the system recognizes that the preposition $PREP_1$ following NP_1 is moved from the relative clause. Acceptability of the structure is tested by implicitly moving back the preposition and the head NP to the relative clause (S-(PP ($PREP_1$ NP_1))).

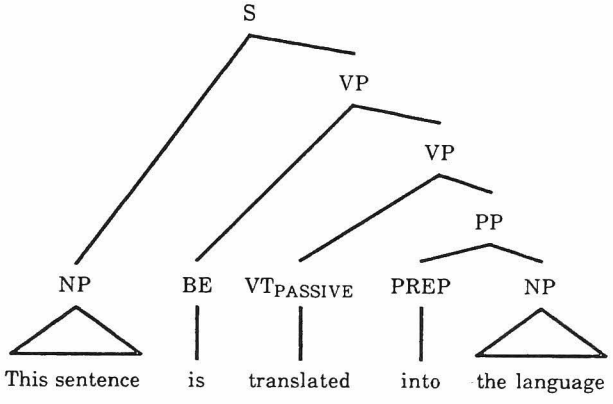
Although, the interpreration of the extended notation for holes is of a rather special type, this mechanism can be used in languages other than English. For example, we can give the same treatment to some classes of embedded sentences in Japanese, as shown in figure 4.2.

A number of constraints are known as to on NP movement[Marcus 80]. Those constraints are incorporated as a procedural rule and will be discussed later.

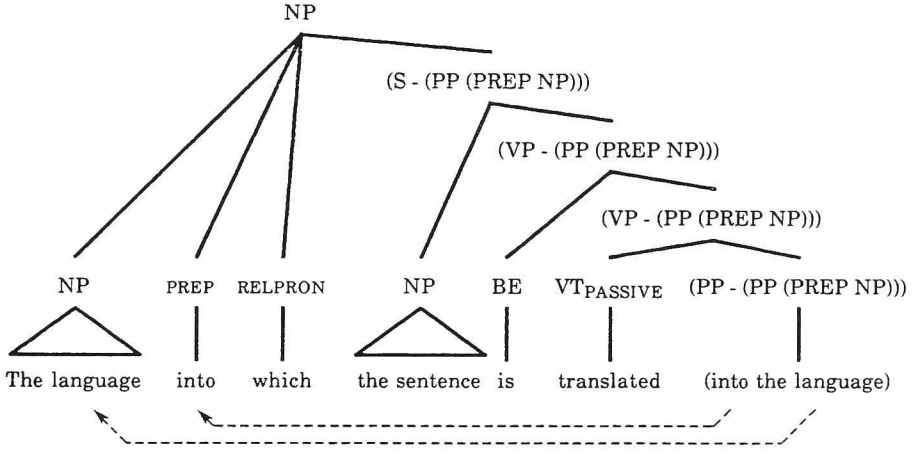
4.4 Procedural Rules

The augmented phrase structure grammar used in describing the structure of language in a declarative manner is not the most appropriate for language analysis. It would be more efficient to give an explicit declaration as to when each rule is to be activated. Additionally, some of the constraints on linguistic constructions are easier to treat procedurally.

We assume a rather general problem solver which will try to find a solution by combining top-down and bottom-up search. We call this general problem solver GPE or General Parsing Engine. We translate augmented phrase structure rules



(a) Phrase Structure Tree for a sentence: "This sentence is translated into the language."



(b) Phrase Structure Tree for a Noun Phrase:
"the language into which the sentence is translated."

Figure 4.1 Augmented Phrase Structures for Holes.

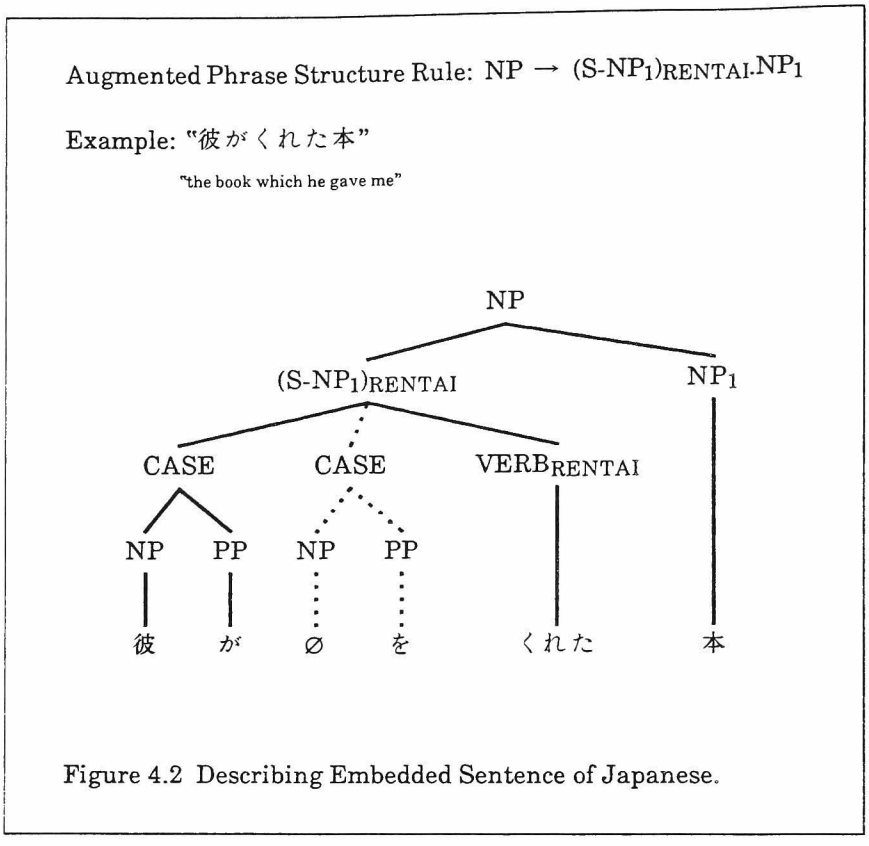


Figure 4.2 Describing Embedded Sentence of Japanese.

into procedural rules which will be activated in each situation of the parsing process.

Procedural rules are realized as a set of situation-action rules. They are classified into four types (E-, U-, B-, L- rules) and will be activated in different situations. When they are activated, only those whose condition matches the input are applied. When there are more than one candidate, they are applied nondeterministically using a backtracking mechanism.

Given an input sentence, GPE will start parsing with the distinguishing symbol S as a goal. GPE uses E-rules to expand a current goal into a set of subgoals. If the input word involves a word specific rule (L-rule), it will also be activated. When a partial parse tree is constructed, U-rules are activated to look for right associative constituents. Using this mechanism we can deal with left recursive structures.

E-rules

The rule format for an E-rule is:

if goal = <goal> **then** ... <condition>$_i$ → <action>$_i$; ... .

An E-rule is activated when a corresponding goal is set up. If condition$_i$ holds for the input, action$_i$ will be applied. When more than one condition holds, all of them will be executed nondeterministically.

Actually, conditions and actions are arbitrary LISP functions. A number of built-in functions are provided to make it easy to translate agumented phrase structure rules into procedural formalism. Table 4.2 illustrates these functions.

Table 4.2 Built-in Functions for Procedural Rules.

name	function
expect[category]	sets up a goal for *category*. If the goal succeeds, a partial parse tree for the category will be pushed into the stack.
pop[n]	pops up *n* partial parse trees from the stack and moves them into the temporal buffer.
construct[category]	builds a partial parse tree of *category* with partial parse trees in the temporal buffer as immediate descendants and with the current set of features as a feature complex of the new node.
getf[feature,n]	gets the value of *feature* of the *n*-th son of the current node.
setf[feature,value]	sets the value of *feature* at the specified value *value*.
reject	terminates the application of the current rule as a failure.
?cat[list-of-categories]	tests whether the current word has a category involved in the *list-of-categories*.
?lex[lexical-item]	tests whether the current word is equal to *lexical-item*.

For example, corresponding to an augmented phrase structure rule:

```
S → NP.VP
where
if inconsistent[NP-VP; list[number[NP]; person[NP]; verb-form[VP]] then reject;
sem = sem[NP](sem[VP]).
```

we can define an E-rule:

```
if goal = S then progn[  expect[NP];
                         expect[VP];
                         pop[2];
                         [inconsistent[NP-VP; list[  getf[NUMBER; 1];
                                                     getf[PERSON; 1];
                                                     getf[VERB-FORM; 2]]]
                                 → reject[]];
                         setf[SEM; list[getf[SEM;1]; getf[SEM; 2]]];
                         construct[S]].
```

We can incorporate an additional test to make the expectation precise. For example, we can make an additional test for prepositional phrase as follows:

```
if goal = PP then ?cat[PREP] → ... expect[PREP]; expect[NP]; ... .
```

Due to the test, the E-rule will be applied only when the current word is of category PREPosition.

U-rules

The rule format for a U-rule is:

if constructed = <category> **then** ... <condition>$_i$ → <action>$_i$; ... .

Like E-rules, conditions and actions are arbitrary LISP functions. The set of built-in functions shown in table 4.2 are available, too.

Generally, left recursive augmented phrase structure rules are translated into U-rules. For example, corresponding to an augmented phrase structure rule:

```
NP → NP.PP where  sem = sem[PP](sem[NP]),
```

we can write a U-rule:

```
if constructed = NP then progn[    expect[PP];
                                   pop[2];
```

```
                    setf[SEM; list[getf[SEM; 2]; getf[SEM; 1]]];
                    construct[NP]].
```

We need a specific procedure for checking well-formedness of relative clauses. Since the procedure depends heavily on the algorithm and the data structure of the parser, we will discuss about that in the next section.

B-rules

The rule format for a B-rule is:

when goal = <category>
  if word-category = <category>
    then ... <category>$_i$ → <action>$_i$; ... .

B-rules add the flavor of the bottom up feature. But, the activation of B-rules is rather restricted; they will be activated only when the partial parse tree is generated by dictionary consultation.

For example, for an augmented phrase structure rule:

```
NP → DET.NOUN
where
if inconsistent[DET-NOUN; list[number[DET]; number[NOUN]] then reject;
number = number[NOUN];
person = person[NOUN];
sem = sem[DET](sem[NOUN]).
```

we can write a B-rule:

```
when goal = NP
  if word-category = DET
    then T → progn[  expect[NOUN];
                     pop[2];
                     [inconsistent[DET-NOUN; list[   getf[NUMBER; 1];
                                                     getf[NUMBER; 2]]]
                          → reject[]];
                     setf[NUMBER; getf[NUMBER; 2]];
                     setf[PERSON; getf[PERSON; 2]];
                     setf[SEM; list[getf[SEM; 1]; getf[SEM; 2]]]
                     construct[NP]].
```

L-rules

L-rules are rules embedded in a dictionary entry. The rule format is:

if goal = <goal> **then** ... <condition>$_i$ → <action>$_i$; ... .

This format is the same as for E-rules. The difference is that an L-rule will be activated only when the corresponding lexical item is involved in the input sentence and when corresponding goal is set up at that position, while E-rules will be always activated when a corresponding goal is set up.

For example, for an augmented phrase structure rule:

NP → both.NP.and.NP where ... ,

we can write an L-rule for the lexical item "both", as follows:

```
both: ...
    L-rule: if goal = NP then progn[   expect["both"];
                                       expext[NP];
                                       expect["and"];
                                       expect[NP];
                                       pop[4];
                                        ...
                                       construct[NP]].
```

4.5 Programming System for Natural Language Analysis

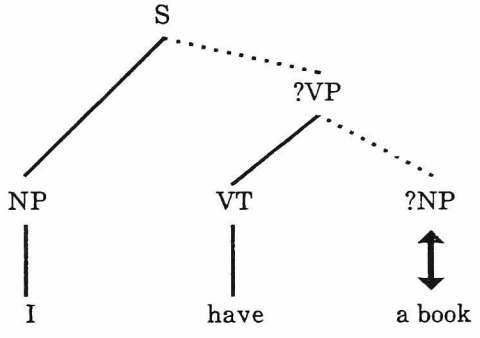
4.5.1 Parsing Algorithm

We use a data structure called packet to keep track of each branch of the nondeterministic application of procedural rules. A packet contains information about the current goal or next instruction, future plan of rule application, current point of the scanner, a stack of partial trees obtained so far, and a list of inherited features. Figure 4.4 illustrates a packet structure for a snapshot of a parsing process.

During parsing a sentence, only one packet is active and others are stored in a stack called *sstack*. As illustrated in figure 4.5, the parsing algorithm repeats a cycle of popping up a packet from sstack, executing the next instruction, and pushing the updated packet and newly created packets into sstack. Parsing is started with an initial symbol S as a goal. Execution of an instruction (SUCCEED)

A:	Current Goal or a Procedure to be executed Next
B:	Stack of Goals or Procedures
C:	Position of the Scanner
D:	Stack of Left Parse Trees
E:	Inheritance Features

(a) Packet Structure.



(b) A Snapshot of a Parsing Process.

A:	NP
B:	((POP 2) (CONSTRUCT VP) (POP 2) (CONSTRUCT S) (SUCCEED))
C:	3
D:	(VT:[...] have NP:[...] I)
E:	NIL

(c) A Packet Describing the Situation of (b).

Figure 4.4 Use of Packets to Describe a Parsing Status.

indicates the parsing is successful. If it is executed, the interactive diagnostic routine will be called to submit the result to the user.

```
parse[buffer; mode]=prog[[count];
                         count ← 0;
                         try[S; 1];
                         print[count; "-parses are found"]]

try[category; position]=prog[[packet; sstack; found-flag];
                         packet ←
                             make-packet[category; ((SUCCEED)); position; NIL; NIL];
                         sstack ← push[packet; NIL];
                         until or[null[sstack]; and[found-flag; mode]]
                             do packet ← top[sstack];
                                sstack ← rest[sstack];
                                try-packet[packet];
                         end;]

try-packet[packet]=prog[[goal; rules; newpackets];
                         goal ← packet.a;
                         newpackets ← ();
                         check-to-see-gap;
                         [atom[goal] →{goal ← list[EXPAND; goal]}];
                         eval[goal];
                         "newly created packets as well as updated packets are set in the variable:
                         newpackets."
                             foreach packet in new packets
                         do sstack ← push[packet; sstack] end
                         ]
```

Figure 4.5 Rough Sketch of Parsing Algorithm.

Each built-in function in procedural rules is actually a procedure which updates the content of the active packet or which will create new packets.

Memorizing Intermediate Results

Each intermediate result during a parsing is saved and will be utilized later when the same goal is set up. The logical structure of intermediate storage is the

same as those used in chart parsers. Unlike chart parsers, however, our intermediate storage does not contain active edges, but it has a mechanism for holding the record of rejected substructures.

Figure 4.6 illustrates an example. Edges with shaded labels (e.g., PP: for writing) are those rejected by the semantic analyzer or the user through interactive diagnosis. Such edges will not be tried again.

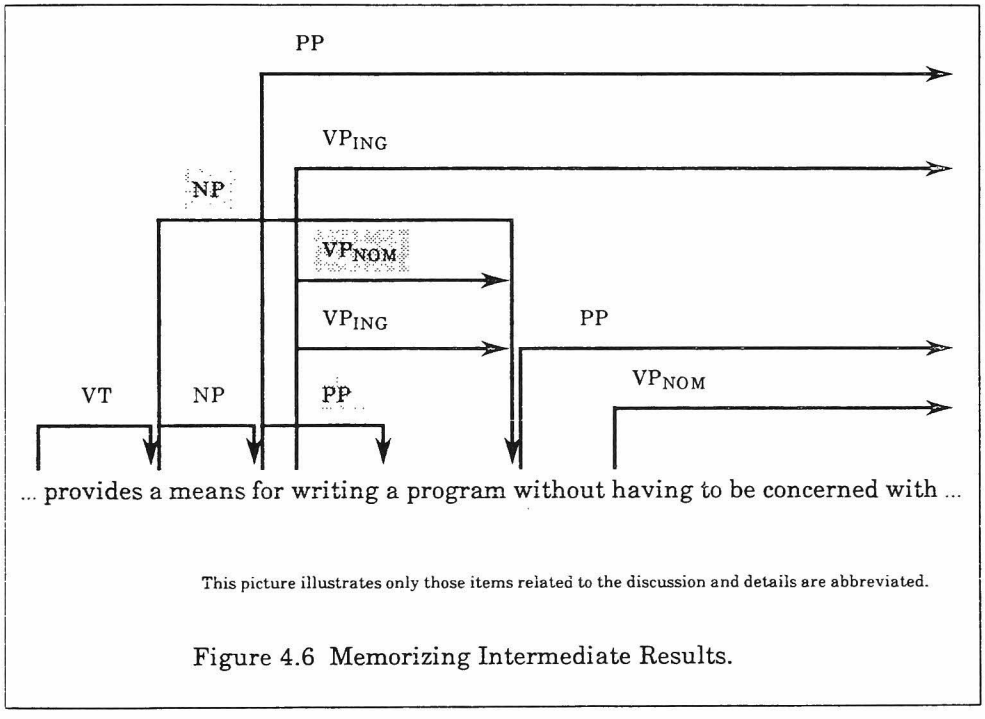


Figure 4.6 Memorizing Intermediate Results.

Completeness of the Algorithm

Apparently, this algorithm is efficient, since it is useful in preventing the same goal from being tried repeatedly. Actually when an input sentence is lengthy, this memorizing greatly improves the inefficiency brought about by a top down backtracking algorithm. But there is a trade off. With memorizing algorithms, we cannot write a radically best first algorithm by rearranging the sequence of rule application in a most-plausible-one-first manner, for completeness may not be assured. Without the memorizing mechanism, no problem arises in such a case, for different packets never affects each other. But with the memorizing mechanism, there can be a side effect. Suppose one rule expands the current goal. When the first possibility succeeds, if another rule submits the same goal, it will immediately

succeed. But its temporary success may be wrong and other possibilities may lead to ultimate success. The problem is that there is no means for the new goal to know all other possibilities. A process which submitted a new goal is given only one solution and that is all. This could be avoided by keeping-tracking complete history of computation, but it seems to complicate the parsing algorithm to a large degree.

The only thing that can be done without spoiling the completeness is to arrange the application order of the rules related to the same goal.

Writing a Procedural Rule for Relative Clauses

In procedural rules, we have to write a more detailed rule to rule out analyses such as:

*a pen which Ø buy Ø (two gaps in a relative clause)

*a pen which he buys it (no gap in a relative clause)

We also work out a mechanism for sending information about the head NP to the gap in a relative clause. We hold information about the head noun in the E-slot of a packet. Using this information, we have to fill the gap by creating a trace node. According to [Marcus 80], *trace can be thought of as a dummy NP that serves us a place holder for the NP that earlier filled that position; in the same sense, the trace can be thought of as simply a pointer to that NP.*

In syntactic analysis, a part of the feature complex for a trace node is not the *same* as that for the head NP, for the syntactic role may differ. For example, suppose a sentence:

"The data is handed to him, who Ø is a scientist."

The head noun of the relative clause is "him". If we simply substituted the head noun into the gap in the relative clause, we would get an unacceptable subexpression:

"*him is a scientist,"

which would cause the parser to reject the whole sentence. Accordingly some attributes of the head noun (surface case inflection, in the above case) should not be copied into a trace node; they have to be modified.

Two built-in procedures are used for declaring a trace node. A procedure dcltrace(x,p) creates a new node by copying the feature complex of the given node x and it invokes a procedure p to modify some attributes of the new node. A procedure *del(x,y) puts a node y into the E-slot of the current packet as a trace node and declares x as a current goal.

To fill the requirement for a relative clause to be well-formed (to have exactly one gap somewhere within its scope), we put a flag in the E-slot to indicate whether or not a gap is ever filled by a trace node. In the beginning, the flag is OFF. The gap can be filled only when the flag is OFF. When a gap is filled by a trace node, this flag is switched ON. A procedure is inserted into the next goal stack (B-slot) of the packet, to check to see that the flag is switched ON after a relative clause is processed. If it is switched ON, the test is successful; otherwise, the analysis fails and other possibilities are sought. Figure 4.3 summarizes this process using an example.

A relative clause may be nested as in a sentence:

"This is a machine that he has developed based on the theory which his professor told him."

In that case, a flag-trace pair is pushed into a stack held in the E-slot of the current packet.

4.5.2 Interactive Diagnosis

The user can tell the system the location where a wrong analysis is made. Usually the output tree is too big for the screen, so a set of commands are provided for moving around on the augmented phrase structure tree and focussing on each part. Table 4.3 illustrates a list of these commands.

For example, as illustrated in figure 4.7, execution of a command:

SCAN (NOUN NOUN NOUN),

will shift the current node pointer to the head of a complex noun which is under the current scope and which matches the specified pattern.

In this example, the new current node pointer will be set to a node governing a word sequence "speed processor".

The user can tell the system that this substructure is inadequate by typing:

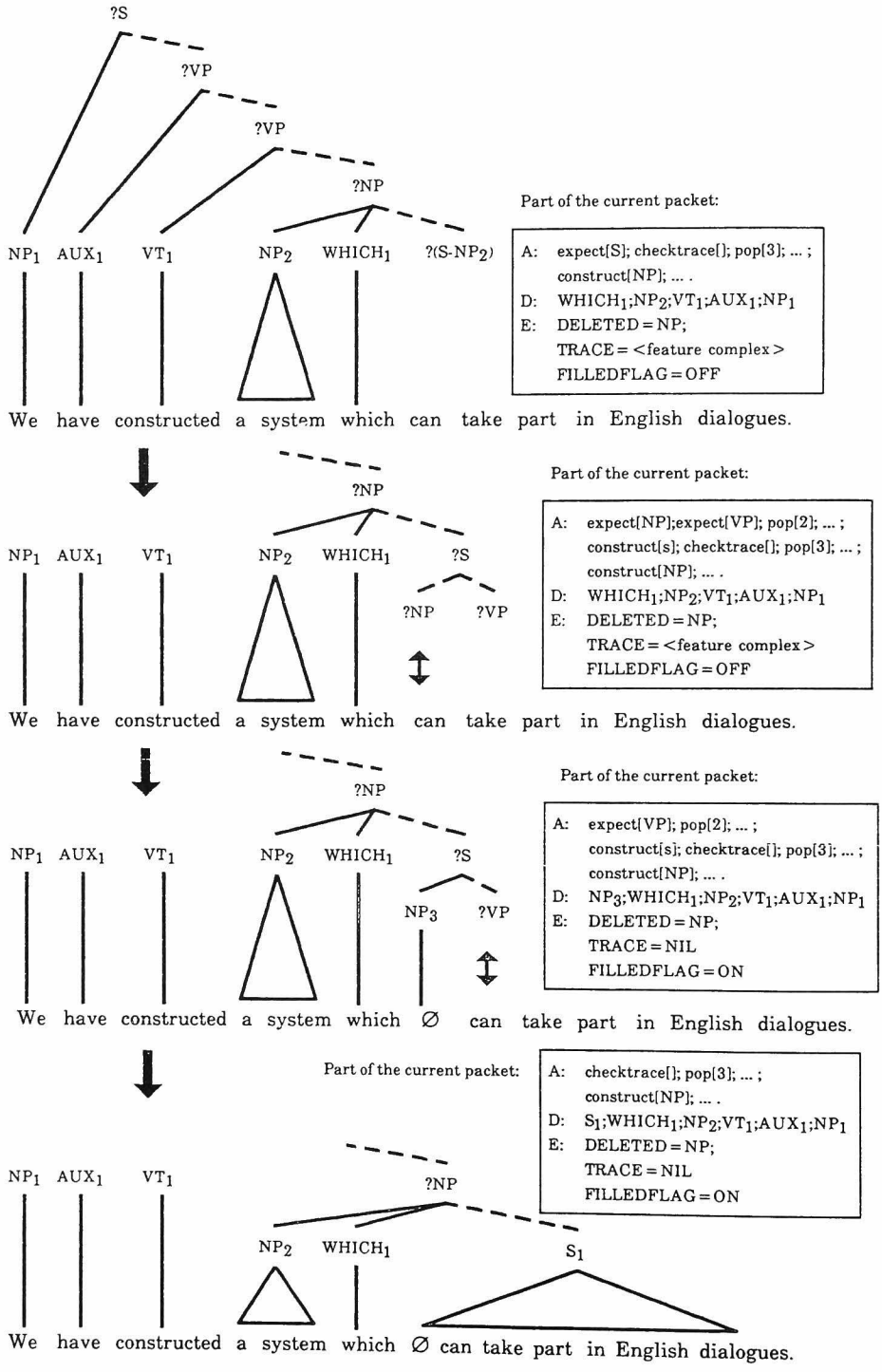


Figure 4.3 Analysis of a Relative Clause with Procedural Rules.

Table 4.3 Commands Used for Interactive Diagnosis.

command	function
SCAN α	traverses all nodes one by one which match α. The scope of traverse is limited to nodes which are (immediate and indirect) descendants of the current node.
F	moves to the next node which matches α in the context of SCAN α.
B	moves to the previous node which matches α in the context of SCAN α.
UP	moves to the parent node.
DOWN	undoes the move by the command UP.
D *n*	displays to the *n*-th level the tree structure governed by the current node.
FEA *f*	displays the value of the feature *f* for the current node; if *f* is *, then it displays all feature values.
SEM	pretty prints the value of *sem* feature of the current node.
TREE	displays the tree structure corresponding to the current node.
GOOD α	tells the system that the node which matches α is adequate; the system will destroy the other node which has intersection with the specified node.
NOGOOD α	tells the system that the node which matches α is inadequate; the system will destroy the node and its ancestors.
ACCEPT	accepts the current analysis, and saves the result.
REJECT	rejects the current analysis; the system will look for other possible analyses under the constraint indicated by GOOD and NOGOOD commands.
EXHAUST	changes the mode to exhaustive search mode.

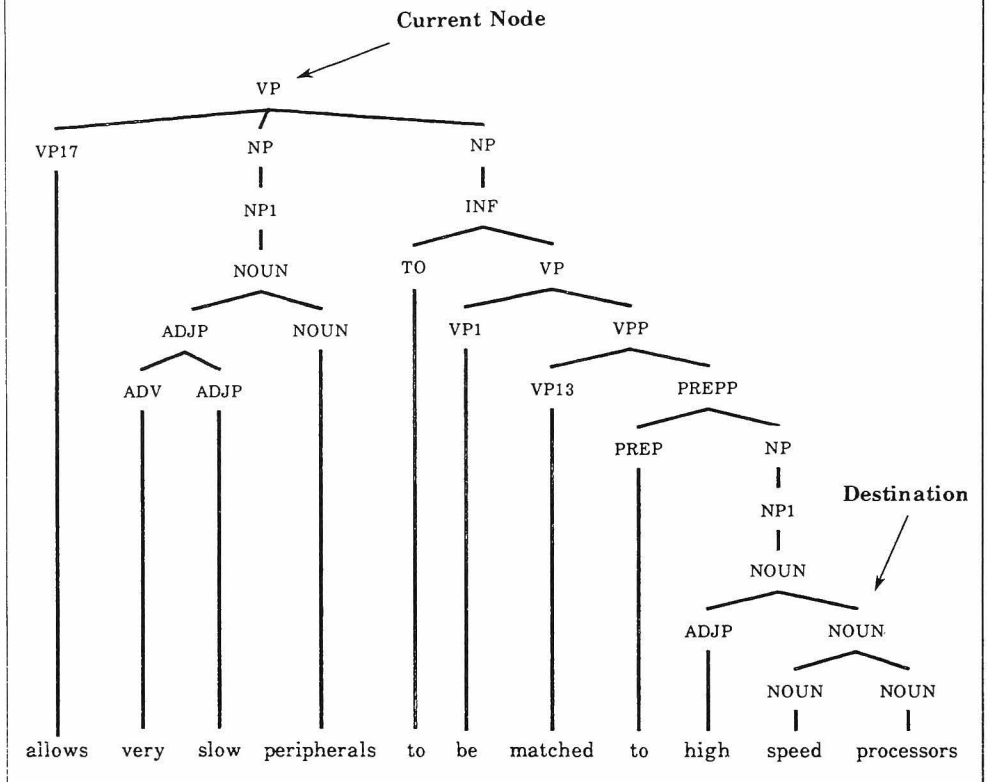


Figure 4.7 Using Diagnostic Commands to Shift Attention.

Executing SCAN (NOUN NOUN NOUN) will shift the attention from the current node to a node which is under the current scope and which will match the given pattern.

NOGOOD *

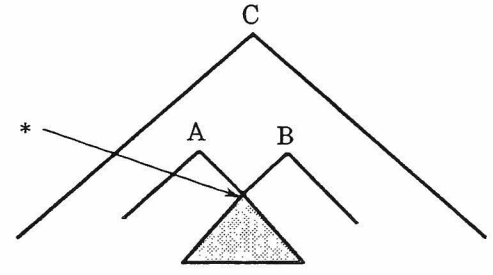
The user could get the same effect by typing a command:

NOGOOD (NOUN NOUN NOUN)

at the initial situation of this example.

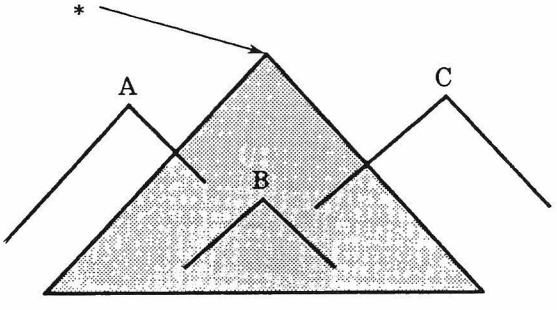
Given diagnostic information about the inadequate analyses, GPE destroys every substructure which has the specified structures as part (figure 4.8a). On the other hand, given information about correct substructures, GPE tries to destroy

every other structure that has intersection with the correct substructures (figure 4.8b).



Substructures A, B, and C will be destroyed.

(a) When the command is NOGOOD *.



Substructures A, B, and C will be destroyed.

(b) When the command is the GOOD *.

Figure 4.8 Scope of Structures to be Destroyed by Diagnostic Commands.

4.5.3 Utilities for Dictionary Management

A simple dictionary editor is developed, which is loaded together with the parser and which will be invoked when an unknown word is encountered or the user wants to update the dictionary.

On the other hand, a simple command language is installed to make a batch type dictionary update. As shown in figure 4.9, the user can use pattern directed descriptions to refer to a class of dictionary entries which he wants to see or modify.

```
<procedure> ::=   ({FOREACH¦FORFIRST¦FORUNIQUE}
                        (PATTERN <pattern>)
                        (IN <scope>)
                        (DO <procedure>)
                        [(CHECK.YES)]) |

                  (IF <conditional expression> TEHN <procedure>
                     [(ELSEIF <conditional expression> THEN <procedure>]*
                     [ELSE <procedure>]) |

                  (PUT <pattern> IN <scope>) |

                  (REMVP <pattern> FROM <scope>) |

                  (PRINT <pattern>)

<scope> ::=       DICT   <variable>

<conditional expression> ::=   (HAS <variable> <pattern>)

<pattern> ::=     <constant> |
                  <variable> |
                  ([<pattern>¦<list variable>]+) |
                  (<variable> WHERE <conditional expression>)

<list variable> ::= (SEG <variable>)
```

Notations $\{\alpha_1 \mid \ldots \mid \alpha_n\}$,$[\alpha]$, $[\alpha]^*$, and $[\alpha]^+$ represent selection from $\alpha_1, \ldots, \alpha_n$, optional selection of α, more than zero repetition of α, and more than one repetition of α, respectively.

Figure 4.9 Part of a Command Language for Dictionary Management.

For example, a command:

```
(FOREACH    (PATTERN (x NOUN (SEG y)))
            (IN.DICT)
            (DO    (IF (AND (HAS y (SEM.α))
                            (NOT (HAS y (AKO.α))))
                       THEN (PUT (AKO.α) IN y))))
```

will add an attribute-value pair: AKO = α for all dictionary entries x such that the category of x is NOUN, and x has an attribute-value pair SEM = α but x does not have AKO = α. For example, a dictionary entry:

(COMPUTER NOUN ... (SEM . COMPUTER) ...)

will be changed as follows:

(COMPUTER NOUN ... (SEM . COMPUTER) (AKO. COMPUTER) ...).

Note that LISP notation is used here.

4.6 Writing Grammar Rules for Analyzing English

In summary, writing rules for translating English into formal representation involves the following steps:

(step 1) Writing a set of rules and a dictionary for associating a piece of formal representation with each phrase structure of English. We can do this using a rule formalism introduced in chapter 3.

(step 2) Augmenting the association rules by adding a number of features to incorporate detailed constraints. For example, features representing person, number or verb form are used to check the subject NP-predicative verb agreement.

(step 3) Attaching control information to each augmented phrase structure rule and transforming it into a procedural rule.

(step 4) Such procedural rules can be improved by incorporating additional constraints or by replacing them with more efficient procedures.

5. Generating Japanese from Formal Representations

5.1 Introduction to chapter 5

This chapter discusses a procedure for generating expressions of Japanese from expressions of EFR. In chapter 3, we described the procedure for creating semantic networks from expressions of EFR. In that process, we first substitute an appropriate semantic network maniputation function into each lexical item of the given EFR expression, and then we evaluate the resulting formula. The evaluation is done in a step-by-step manner, combining pieces of semantic networks together. We can obtain the syntactic structure of a target language as a result of the evaluation. The higher order property of EFR is inherited by CPSF. This brings us what we call the higher order problem when we generate a structure from expressions in EFR. We employ heuristic solutions to solve this problem. Figure 5.1 shows the system organization for the generation stage.

5.2 Formal Tools for Generation

5.2.1 CPS

We use an annotated phrase structure called CPS (Conceptual Phrase Structure) to represent the syntatic structure of Japanese. Like syntactic structures for English, each node of a structure of CPS is given a set of attribute-value pairs to provide additional information involving semantic class and other semantic characteristics of the node. A definition of the structure of CPS is given below:

CPS : [$_{\text{Category}}$ Descendants **with** AttributeValuePairs]

In defining the syntax of CPS or CPSF below, gothic letters are used to denote constants. Other symbols denote metavariables.

Descendants : a sequence of CPS or a terminal symbol; a terminal symbol is delimited by double quotations.

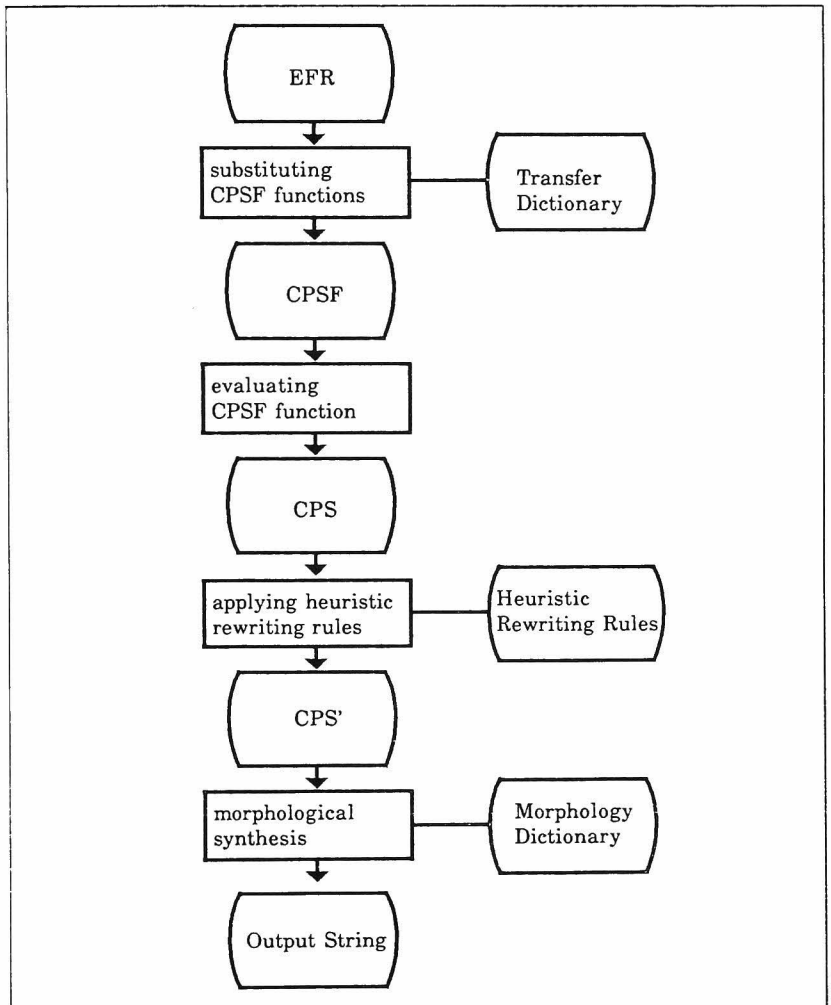


Figure 5.1 Module Organization for the Generation Stage.

AttributeValuePairs: a sequence of the form: Attribute= Value.

(Examples)

[NP "英語" with class=abstract, isa=language, count=unable]
"English"

[S [CASE [NP "計算機"] [PP "が"]]
[CASE [NP "自然言語"] [PP "を"]]

[PRED [VERB "理解する"]]]

"The computer understands natural language"

We often use the same kind of graphical notations as augmented phrase structures for English, to make it easy to grasp the overall configuration of CPS. Figure 5.2 illustrates graphic notations for the above CPS structures.

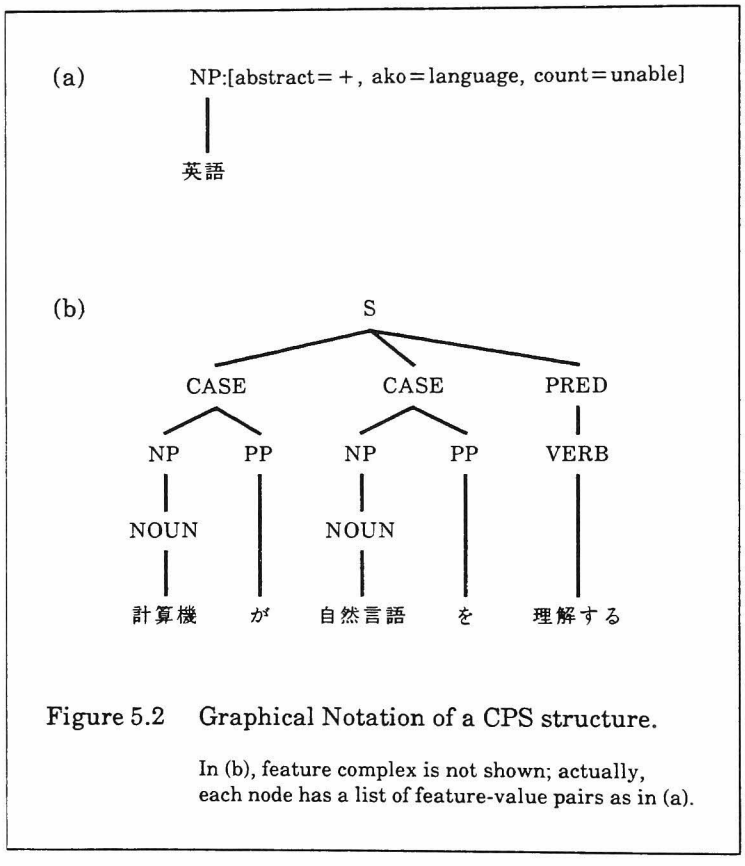


Figure 5.2 Graphical Notation of a CPS structure.

In (b), feature complex is not shown; actually, each node has a list of feature-value pairs as in (a).

5.2.2 CPSF

CPSF is a functional language for denoting operations on CPS. Thus operation is defined in functional application forms. Below are shown constituents of CPSF with some examples.

Constants

CPSF involves structures of CPS as constants.

Variables

x, y, z, ...

Variables with Constraint

!Category(Variable)

The range of the variable should be a structure of CPS in the indicated category; otherwise, a procedure for coercion will be called to try to transform the value into a structure of CPS. If this is not successful, error will be announced.

Lambda Formulas

λVariableList[LambdaBody]

When this is applied to a sequence of arguments, each variable in VariableList will be bound to the argument at the corresponding position and LambdaBody which in turn is a CPSF formula will be evaluated in the resulting environment.

Compositions

{$_{\text{Category}}$ Descendants **with** AttributeValuePairs}

Conditionals

{$\text{Condition}_1 \rightarrow \text{CPSF}_1$; ... }

e.g., break : λxy[{class(x) = animate → {S ... xがyを壊す ... }; "x breaks y"

else → {S ... xのせいでyが壊れた ... }}] "y is broken due to x"

Transformation

+ $\text{TransformationName}_{\text{ParameterList}}$(Argument)

e.g., $+\text{TENSE}_{\text{TENSE}=\text{PAST}}$(x).

We can give a definition using lambda formula. Thus,

$+\text{TENSE}_{\text{TENSE}=\text{PAST}} \equiv \lambda x[\{S\ \ !S(x),\ [\text{AUX}\ \ \text{"た"}]\}]$.

5.2.3 Evaluation of CPSF formulas

In evaluating functional formulas of CPSF, a problem called the higher order problem arises.

Higher Order Problem

Expressions in EFR have higher order property. By higher order property we mean that functions may be modified by other functions. Making modification to functions is a very complicated task; we need to analyze the structure of the function as well as to determine the output form. We call this problem the *higher order problem.* Since the formulas of CPSF are obtained from expressions of EFR, they also have this property.

For example, EFR expressions for adjectives are operators to nouns, such as:

large(database).

Since the Japanese translation of English adjectives are also modifiers to noun, we could make an assignment as follows:

large ← λx[{NOUN [ADJ "大きな"],x }].

When applied to a CPS of category NOUN, this lambda formula will attach a CPS of category ADJ to the argument, resulting in a complex noun (category NOUN again).

On the other hand, adjectives can be modified by adverbs, such as:

extremely(large).

The translation of this EFR should be of the same category as the simple adjective "large", such as:

extremely(large) : λx[{NOUN [ADJ [ADVDEG "極めて"],[ADJ "大きな"]],x }].

The difficulty here is in the formula of CPSF to be assigned to the adverb "extremely" itself. It must be something which will transform a lambda expression into another lambda expression. Doing this kind of task in general is very complicated and inefficient.

Basically, the higher order property of EFR or CPSF can be thought of as only a notational problem, so that we can avoid much of the difficulty.

In the above example, we can avoid this problem by simply assigning a CPS structure to each adjective. For example,

large ←[ADJ "大きな"].

In doing so, there will arise a situation where this CPS comes in the operator position:

[ADJ "大きな"]([NOUN "建物"]).
"big" "building"

In such a case, we have to define what to do. Application rules are used for such purposes. In this example, an *application rule* looks like:

[ADJ x]([NOUN y]) ⇒ [NOUN [ADJ x],[NOUN y]].

Unfortunately, this does not solve the problem completely. There remains a possibility of lambda's arising from lambda forms of EFR. For example, an expression of EFR for a complex noun:

"block on the table"

looks like:

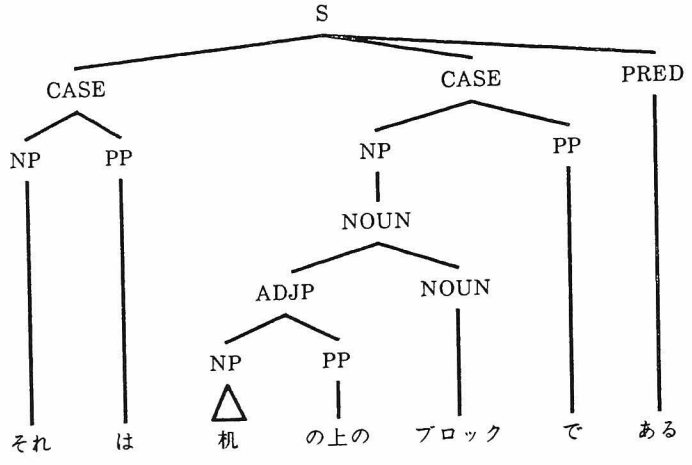
λx[(the(table))(λy[(((*ap(on))(y))(block))(x)])].

In the transfer stage, we substitute formula of CPSF into each lexical item of this expression, and this will result in another lambda formula of CPSF:

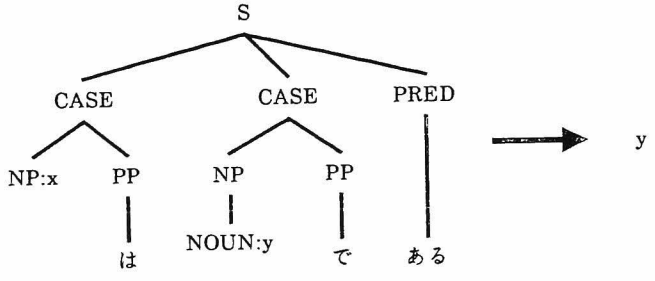
λx[(the*(table*))(λy[(((*ap*(on*))(y))(block*))(x)])],
where α* (α= the, table, ...) represents CPSF formula substituted into α.

The problem is that there is a possibility in the evaluation stage that the evaluator has to modify the lambda formula into another structure. It is too difficult to directly carry out transformation of lambda formulas in such cases.

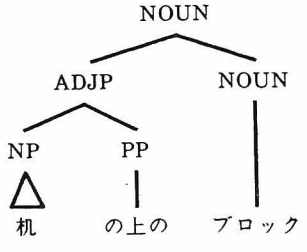
Fortunately, we can know the type of each lambda formula. The type of the lambda formula above, for example, is a one-place predicate. Since a one-place predicate is a function taking an individual and resulting in a proposition, we will get a sentence structure if we apply the lambda formula to a known individual, say "それ" (it). From the resulting sentence structure, we can infer a structure of CPS which is equivalent to the given lambda formula. Figure 5.3 illustrates how our example is treated. We call rules like the one used in this example *coercion rules.*



(a) Sentence Structure Obtained from Lambda Application.



(b) Inference Rule (Coercion Rule).



(c) A CPS Structure Equivalent to the Given Lambda Formula.

Figure 5.3 Inference of CPS Structure Equivalent to a Given Lambda Formula of CPSF.

Of course, this kind of processing is not desirable, as it introduces an extra complexity. But this is a tradeoff resulting from employing formal semantics to achieve systematicity.

Evaluating Basic Formulas of CPSF

An algorithm for evaluating formulas of CPSF is almost the same as a LISP interpreter. The value of a variable is accessed through an association list. As to lambda conversion, delayed evaluation is used. A FUNARG triple is created for each functional value.

As to transformation functions, the grammar designer has to directly write a LISP procedure to make the intended transformation. Although this is inconvenient, we have found that most transformations can be replaced by more systematic operations such as application rules.

5.3 Generating the Syntactic Structure of Japanese

5.3.1 Categories of CPS and CPSF

The first thing to do in designing CPSF functions for generating syntactic structures of the target language is to determine the set of categories to be used in CPS and CPSF. Table 5.1 shows a list of categories used here and their relation to types of EFR.

CPSF formulas corresponding to the same type of EFR must have the same effect in functional evaluation. For example, both CPSF functions of category noun and that of category intransitive verb must be those that map CPS structures of category noun phrase into CPS structures of category sentence. The following notation represents this convention:

Noun, IntransitiveVerb ≡ Sentence/NounPhrase.

This characteristic is determined according to the facts that:

(a) in EFR, a type $<0,1>$ stands for functions from type 1 to type 0,

(b) CPSF functions of type noun and those of type intransitive verbs can be both substituted into lexical items of type $<0,1>$ of EFR,

(c) CPSF functions of category noun phrase and category sentence correspond to EFR expressions of type 1 and 0, respectively.

Table 5.1 Categories and their Relation to Types of EFR

Type of EFR	Category of CPS	Mutual Relationships among CPSF Categories
0	S (Sentence)	
1	NP (Noun Phrase)	
<0,1>	NOUN (intransitive) VERB	S/NP
<0,1, ... , 1>	VERB	S/NP× ... × NP
<0,<0,1>>	$NP_{FUNCTION}$	S/(S/NP)
<<0,1>,<0,1>>	ADJP (Adjectival phrase)	NOUN/NOUN
<0,0>	ADVP (Adverbial phrase) AUX (auxliary verb)	S/S
< α , α >	ADVDEG (adverbial phrase representing degree)	α / α
<<<0,1>,<0,1>>,1>	MKADJPP (postposition making adjectival phrase)	ADJP/NP
<<0,0>,1>	MKADVPP (postposition making adverbial phrase)	ADVP/NP

Thus categories of CPS (and CPSF) are mutually interrelated. The rightmost column of table 5.1 describes those relationships among categories. Each function must be designed according to this convention.

Exceptional treatment is given to verbs. We do not make subcategorization of verbs, so that intransitive verbs, transitive verbs, etc are given the same category name VERB, even though their functional behaviors are different. In practice, however, this does not raise any problem.

From the viewpoint of Montague grammar, the treatment of noun phrase as being associated with EFR of type 1 might be curious; they might be associated with EFR of type <0,<0,1>>. Our intention is to interpret EFR expressions of type 1 as individuals and to refer to them by using noun phrases. EFR expressions of type <0,<0,1>> are interpreted merely as functions of one place predicates and as a result noun phrases will be generated. Thus, the association of EFR expressions of type <0,<0,1>> with noun phrases is indirect.

5.3.2 Generating from Simple Sentences

A simple sentence of English consists of an NP as subject and an VP as predicate. An EFR expression for an NP is of type $<0,<0,1>>$ and that for a VP is of type $<0,1>$. In CPSF assigment, an EFR expression of type $<0,<0,1>>$ will be assigned a function which takes a one place predicate and which will produce a CPS of category sentence, and an EFR expression of type $<0,1>$ will be assigned a function which takes a CPS structure of category noun phrase and which will produce a CPS of category sentence. Conceptually, the former function will attach case information to the argument, as is shown in figure 5.4a. To be more precise, the argument is a one place function and case binding is done rather implicitly using lambda conversion (figure 5.4b). An application rule used there is:

$$[\mathrm{NP}_{\mathrm{FUNCTION}}\ \mathbf{x}](y) \Rightarrow y([\mathrm{NP}\ \mathbf{x}]).$$

Rules of this type are sometimes called sublimations[Dowty 81].

Dealing with Prenominal Negations

This treatment of noun phrases gives an elegant solution to the prenominal negation problem: if a noun phrase governed by a verb phrase contains constituents like "no", "little", or "few", the whole sentence must be translated into a negative sentence of Japanese, for Japanese syntax does not have a device for prenominal negation. In our solution, an EFR expression for a noun phrase is given a wider scope than a verb phrase governing it, so that we only have to assign a CPSF function containing a negation operator to a noun phrase containing a prenominal negation. For example,

$$\text{"no student"} \Rightarrow \lambda p[+\mathrm{NEG}(p([_{\mathrm{NP}}\ \underset{\text{"some student"}}{\text{"一人の学生も"}}]))],$$

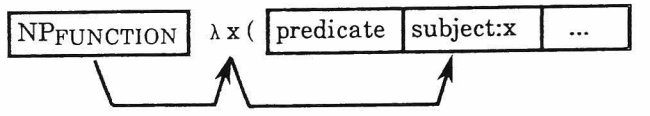
where +NEG is a transformation operator which will make a negation out of a given sentence. Using this CPSF, a sentence:

no student comes

will be assigned a following CPSF function:

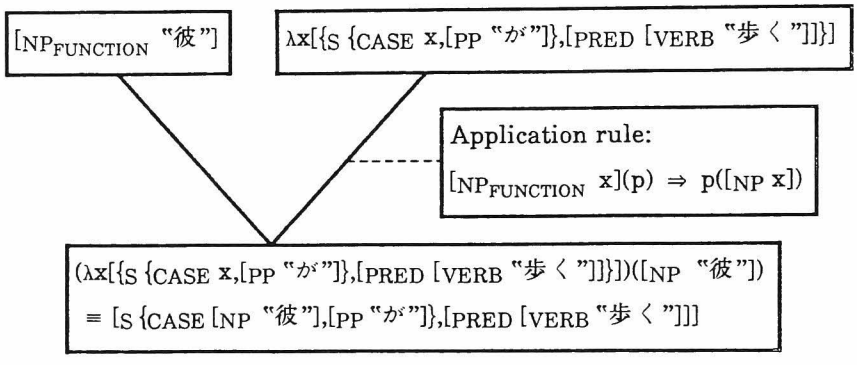
$$(\lambda p[+\mathrm{NEG}(p([_{\mathrm{NP}}\ \underset{\text{"some student"}}{\text{"一人の学生も"}}]))])(\lambda \mathbf{x}\{_{\mathrm{S}}\ \underset{\text{"x comes"}}{\text{"xが来る"}}\}),$$

which will be evaluated to:



sending information about subject case

(a) Basic Idea



(b) Detailed Description: (example).

Figure 5.4 Lambda Application for Sending Case Information from Noun Phrase to Verb Phrase

+NEG((λx{S "xが来る"})([NP "一人の学生も"]))

"x comes" "some student"

⇒ +NEG([S "一人の学生も来る"])

"some student comes"

⇒ [S "一人の学生も来ない"].

"it is not the case that any student comes"

Below are shown other cases where prenominal negation takes place with a transitive verb. Note that we need not change the assignment to the noun phrase containing prenominal negation in each case.

(case 1) noun phrase containing a prenominal negation is at subject position:

Input sentence:

"no student has a text book"

CPSF function to be assigned:

$$(\lambda p[+\mathrm{NEG}(p(\{_{\mathrm{NP}}\ \underset{\text{"some student"}}{\text{"一人の学生も"}}\}))])$$

$$(\lambda x[[\mathrm{NP}_{\mathrm{FUNCTION}}\ \underset{\text{"a textbook"}}{\text{"教科書"}}](\lambda y[\{_{\mathrm{S}}\ \underset{\text{"x has y"}}{\text{"xがyを持つ"}}\}])])$$

⇒... ⇒[S "一人の学生も教科書を持たない"].
"it is not the case that any student has a textbook"

(case 2) noun phrase containing a prenominal negation is at object position:

Input sentence:

"The company employs no student"

CPSF function to be assigned:

$$[\mathrm{NP}_{\mathrm{FUNCTION}}\ \underset{\text{"the company"}}{\text{"その会社"}}]$$

$$(\lambda x[(\lambda p[+\mathrm{NEG}(p(\{_{\mathrm{NP}}\ \underset{\text{"some student"}}{\text{"一人の学生も"}}\}))])$$

$$(\lambda y[\{_{\mathrm{S}}\ \underset{\text{"x employs y"}}{\text{"xがyを雇う"}}\}])])$$

⇒... ⇒[S "その会社が(は)一人の学生も雇わない"]
"it is not the case that the company employs any student"

5.3.3 Generating from Verb Phrases

The center of a verb phrase is a verb. EFR expressions for verbs are of type $<0, 1, \ldots, 1>$. A CPSF function producing a CPS structure of category sentence out of n CPS structures of category noun phrase will be substituted into a lexical item of EFR corresponding to a verb.

For example, to a lexical item "construct" of EFR, a two-place function will be substituted:

$$\lambda xy[_{\mathrm{S}}\ [_{\mathrm{CASE}}\ x, [_{\mathrm{PP}}\ \text{"が"}]]\ [_{\mathrm{CASE}}\ y, [_{\mathrm{PP}}\ \text{"を"}]]\ [_{\mathrm{PRED}}[_{\mathrm{VERB}}\ \text{"作成する"}]]]].$$

Given a pair of CPS structures of category noun phrase:

[NP "私"] and [NP "そのシステム"],

the above function will generate a CPS structure of category sentence:

[S [CASE [NP "私"] [PP "が"]]
[CASE [NP "そのシステム"][PP "を"]]
[PRED [VERB "作成する"]]]

Graphic notation of the result is shown in figure 5.5.

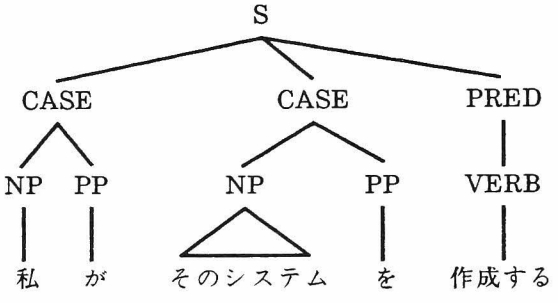


Figure 5.5 CPS Structure of Category S.

The simplest construction of a verb phrase is a verb plus noun phrases. Figure 5.6a gives a basic idea of how CPS structures are generated from EFR expressions for verb phrases of English. Figure 5.6b illustrates a more technical aspect of the treatment.

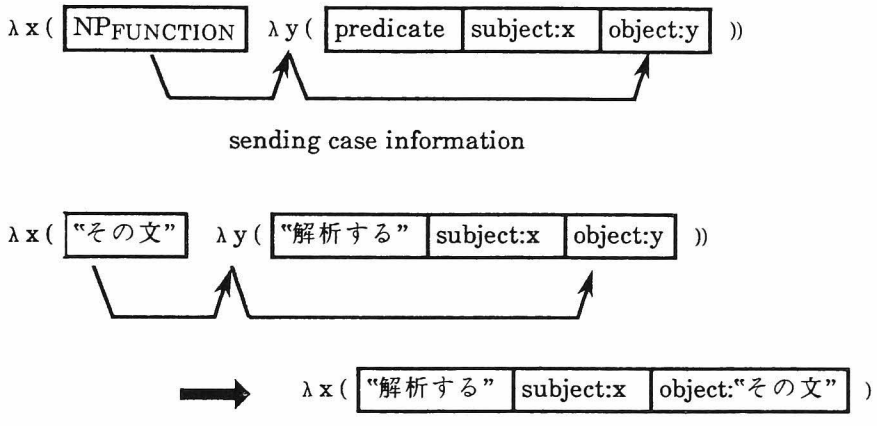
5.3.3.1 Word Choice and Structural Transfer

Using features and tests, we can translate a verb differently according to the case values. For example, the assignment:

develop ← λxy{ ako(y)=film → {S "xがyを現像する"};
else → {S "xがyを開発する"}}

and use of features in noun phrases:

a system : [NP "システム" with ako=system, ...]



(a) Basic Idea.

input: "analyze the sentence"

EFR: λx[(the(sentence))(λy[analyze(x,y)])]

replacement by CPSF function:

the(sentence) ⇒ [$NP_{FUNCTION}$ "その文"]

analyze ⇒ λxy[{S "xがyを解析する"}]

result of substitution:

λx[[$NP_{FUNCTION}$ "その文"](λy[(λxy[{S "xがyを解析する"}])(x,y)])]

≡ λx[(λy[{S "xがyを解析する"}])([NP "その文"])]

≡ λx[{S "xがその文を解析する"}]

(b) Detailed Description: (example).

Figure 5.6 Sending Case Information to the Main Verb.

a film : [NP "フィルム" with ako=film, ...]

will enable the following word choice:

develop+film ⇒ フィルムを現像する

develop+system ⇒ システムを開発する.

As discussed in chapter 2, however, we are not involved in the problem of how to choose a specific set of features and tests for making word choice. Rather, we will be involved in designing an external procedure for allowing the features and tests mechanism to apply to each syntactic variation of a standard form, such as passive sentences or relative clauses, in which the deep case value of a verb is moved from its standard surface position.

5.3.3.2 Passives

In EFR expressions for passive construction, the following operators are used:

*en : of type <<0,1>,<0,1,1>>

*en3 : of type <<0,1,1>,<0,1,1,1>>.

CPSF functions assigned to these operators try to transform a given verb into a passive form by supplying information about the deep subject.

In order to do this, we introduce two devices: a transformation operator +PASSIVE and a special entity COMP*. +PASSIVE transforms a given sentence into passive structure by moving an NP of subject case or of agent case (figure 5.7).

Unlike other entities of CPSF, the value of COMP* depends on context. When there is no context at all, the value of COMP* is simply a null NP:

[NP ""],

while, when there is a context, COMP* can receive a feature complex from the context:

[NP "" <feature complex received from the context>].

A CPSF function assigned to a preposition indicating a deep subject (*psubj) can put conceptual information about the deep subject in context. Thus in the case of

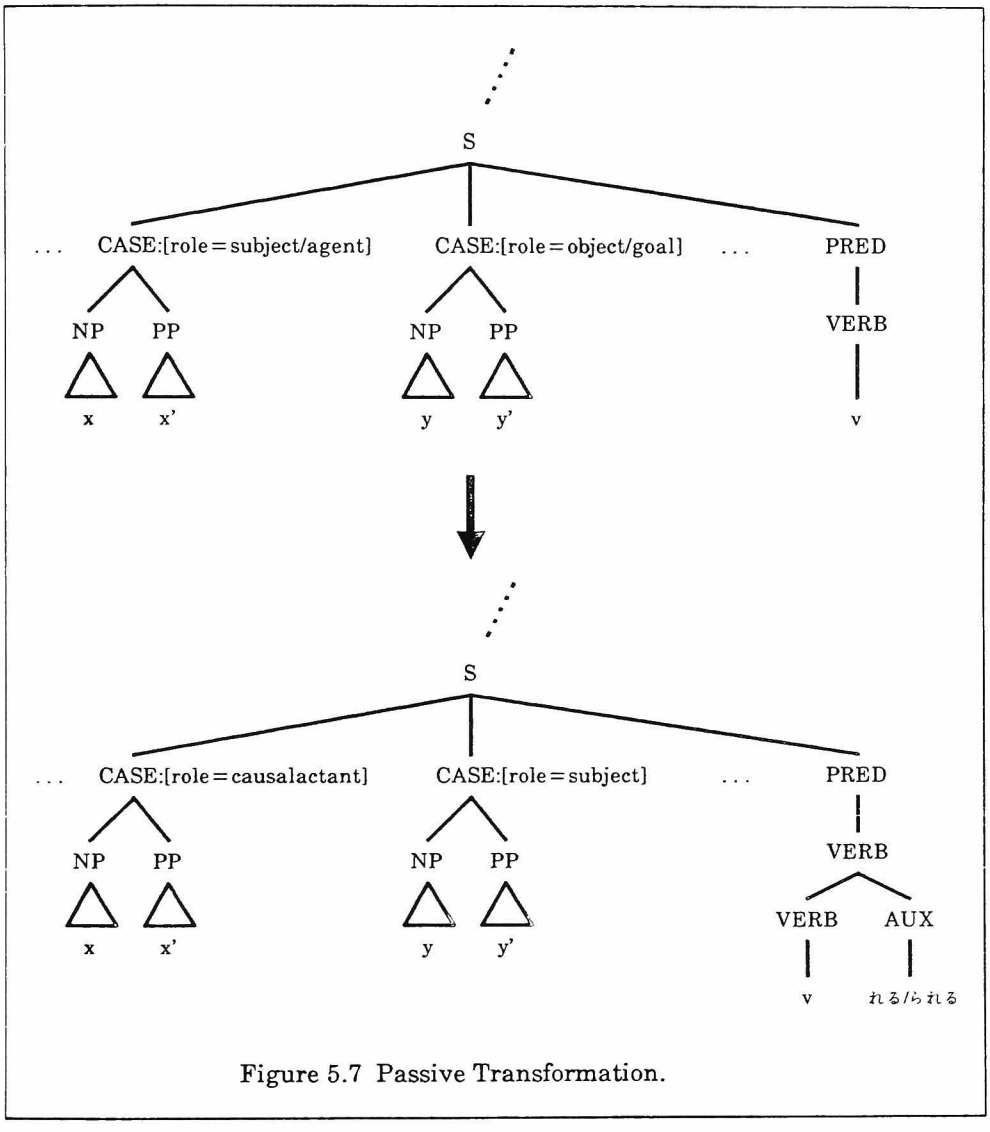


Figure 5.7 Passive Transformation.

passive construction, linking between the deep subject and a verb is done implicitly.

In summary, we make the following assignments:

en ← λv[λx[+PASSIVE(v(COMP,x))]],

en3 ← λv[λxy[+PASSIVE(v(COMP,x,y))]],

*psubj ← [MKADVPP "によって" when-applied:put-the-argument-in-context],

and application rules:

[MKADVPP x]([NP y]) ⇒ [ADV [NP y][MKADVPP x]],

[ADV x]([S y]) ⇒ [S [ADV x][S y]].

(Example) "The window is broken by him."

EFR: he(λx[(*psubj(x))((the(window))(*en(break)))])

Evaluation of a formula resulting from CPSF assignment goes as follows:

(step 1) A variable x is bound to a CPS structure:

[NP "彼" with class=animate].

(step 2) Evaluating a CPSF formula assigned to *psubj(x), we get

[ADV [NP "彼"][MKADVPP "によって"]].

At the same time, conceptual information: class=animate is put into context.

(step 3) Evaluating a CPSF formula for the(window), we get:

[NP$_{FUNCTION}$ "その窓"].

(step 4) Evaluating a CPSF formula for

(*psubj(x))((the(window))(*en(break))),

we get an intermediate formula:

[ADV [NP "彼"][MKADVPP "によって"]]
(+PASSIVE
(break*([NP "" with class=animate], [NP "その窓"]))),

where, break* stands for a CPSF function substituted into a lexical item "break". Note that conceptual information about the deep subject and deep object are passed to a CPSF function break* correctly, making it possible for the function to choose an appropriate word.

(step 5) Finally, we get:

[S [ADV [NP "彼"][MKADVPP "によって"]]
[S [CASE [NP "その窓"][PP "が"]]
[PRED [VERB [VERB "壊す"] [AUX "れる/られる"]]]]].

Graphic description of the result is shown in figure 5.8.

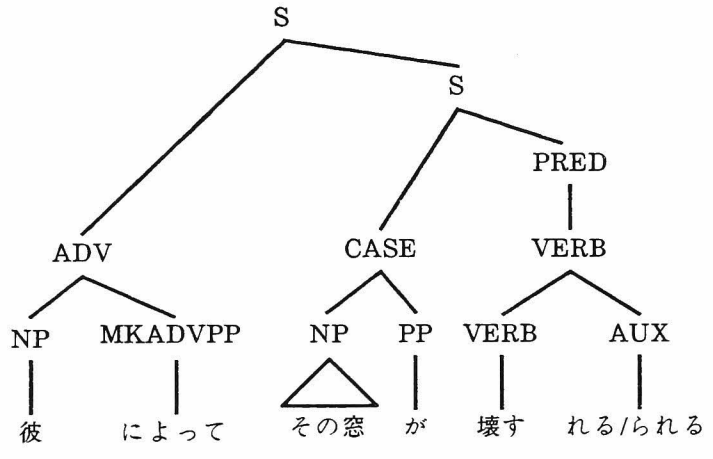


Figure 5.8 A Sentence Strcuture Generated from the EFR Expression for Passives.

5.3.4 Generating from Simple Noun Phrases

The form of EFR expression associated with a simple noun phrase of English is:

<DET>(<NOUN>),

where <DET> and <NOUN> stand for an EFR expression for a determiner and a noun, respectively. A noun involved in a noun phrase may be further modified by a series of noun modifiers. Their treatments are described in the next section.

5.3.4.1 Nouns

We assign a CPS structure of category noun to each lexical item for noun. Thus,

book ← [NOUN "本"].

Since the category noun is displayed as S/NP or a mapping from NP to S, we have to define an application rule for the case in which a CPS structure of category noun is applied to another CPS structure of category NP. The application rule in this case is:

[NOUN x]([NP y])

⇒ [S [CASE [NP y] [PP "は"] with ROLE=SUBJECT]
[CASE [NP [NOUN x]] [PP "で"] with ROLE=PARTICIPANT]
[PRED [VERB "ある"]]].

Thus, a noun x will create a sentence "y is (an) x" when it is applied to a noun phrase y.

5.3.4.2 Determiners

The CPSF function assigned to an EFR expression of category determiner will generate a CPSF function of category $NP_{FUNCTION}$ out of a CPSF function of category noun. In the simplest case, the argument is a CPS of category noun (e.g., [NOUN "本"]). But the argument may be arbitrarily complicated. For example, an EFR expression for a compound noun "(the) length of the ship" looks like:

λy[(the(ship))(λx[((((*ap(of))(x))(length))(y))])].

A lambda form resulting from substituting an appropriate CPSF function into each lexical item in this form will be handed to the CPSF function for a determiner.

Accordingly, we first transform a given lambda form into an equivalent CPS of category noun then we will attach an appropriate determiner. In order to do this, we use a coercion rule:

!NOUN(x): if x is a CPS structure of category NOUN, then return;
else apply x to a dummy NP [NP "それ"] and extract from the result a CPS of category noun using a rule shown in figure 5.9.

This coercion rule will transform the above lambda form into an equivalent CPS structure of category noun:

[NOUN [ADJP [NP "その船"] [PP "の"]] [NOUN "長さ"]].
"the ship" "of" "length"

For a determiner like "a" or "the", the CPSF function assigned is:

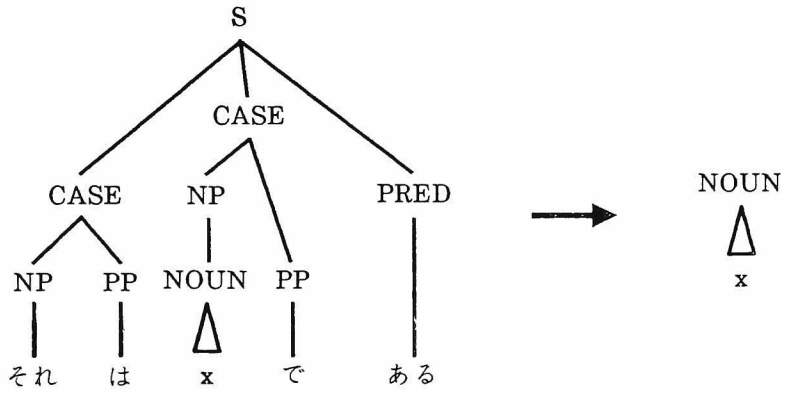


Figure 5.9 A Rule for Extracting a NOUN from a Sentence.

$\lambda p[\{NP_{FUNCTION}$ [DET "その"], !NOUN(p)}].

In the case of a determiner "no" or others which entails prenominal negation, we assign a slightly more complicated function of CPSF, which contains a negation operator to a sentence. Thus,

no ← λp[λq[+NEG(q(+も([DET "一の"](!NOUN(p)))))]]

5.3.4.3 Possessives

The type of EFR expressions for possessives such as "my", "your", "the station's", or "the members'" is the same as that for determiners. Thus a CPSF function assigned to a possessive must be something which produces a CPS of category $NP_{FUNCTION}$ out of a CPSF of category noun.

The form of EFR expression for a substructure:

<NP> 's <NOUN>

is:

λp[<NP>(λx[((*poss(x))(<NOUN>))(p)])].

In order to deal with possessives, we introduce new application rules:

[POSSPP x]([NP y]) ⇒ [POSS [NP y][POSSPP x]],

[POSS x]([NOUN y]) ⇒ [$NP_{FUNCTION}$ [POSS x][NOUN y]].

Note that in the second rule, if the argument is not a CPS structure but is an unevaluated functional value of CPSF, a coercion rule introduced so far is automatically applied to it to transform it into an equivalent CPS structure of category NOUN.

Based on these rules, we make assignment as follows:

***poss** ← [POSSPP "の"].

(Example) An EFR expression associated with a noun phrase:

"a station's facility"

is:

λp[(a(station))(λx[((*poss(x))(facility))(p)])].

Evaluation of a CPSF formula resulting from CPSF assignment is as follows:

(step 1) A variable x is bound to a CPS: [NP "(ある)ステーション"].

(step 2) Evaluating a CPSF formula for a subexpression *poss(x), we get:

[POSS [NP "(ある)ステーション"][POSSPP "の"]].

(step 3) Evaluating a CPSF formula for a subexpression

(*poss(x))(facility),

we get:

[$NP_{FUNCTION}$ [POSS [NP "(ある)ステーション"][POSSPP "の"]]
[NOUN "設備"]].

5.3.5 Generating from Noun Modifiers

5.3.5.1 Adjectives

Adjectives are assigned CPSF functions of category adjective. Since the category adjective is displayed as noun/noun, we need an application rule as follows:

[ADJ x]([NOUN y]) ⇒ [NOUN [ADJ x] [NOUN y]].

5.3.5.2 Adjectival Prepositional Phrases

We analyzed prepositional phrases as modifying phrases to a noun, and we associated with a noun-modifying prepositional phrase an EFR expression as follows:

λx[λn[<NP>(λy[(((*ap(<PREP>))(y))(n))(x)])]].

<NP> and <PREP> represent EFR expressions associated with a noun phrase and preposition, respectively.

In order to generate from this expression a CPS structure of Japanese, we introduce new application rules as follows:

[MKMKADJPP x]([MKADVPP y]) ⇒ [MKADJPP y'],

where y' is an adjectival phrase making a post position corresponding to y; for example if y is "のために" then y' is "のための", if y is "なしに" then y' is "なしの", etc.

[MKADJPP x]([NP y]) ⇒ [ADJ [NP y] [MKADJPP x]].

For each <PREP>, we simply substitute a CPS structure of category MKADVPP. Any lexical item of this category will be combined with a noun phrase and produce a sentence modifier (category ADV). An operator "*ap" is assigned a CPS structure of category MKMKADJPP. According to the first application rule introduced in this section, it will change the category of CPS structure assigned to a preposition from MKADVPP to MKADJPP by replacing the lexical item. The second rule specifies that if a CPS structure of category MKADJPP (adjective making postposition) is applied to a CPS structure of category noun, then it will be attached to the right of the noun and it will result in a CPS structure of category ADJ as a whole.

In summary, CPSF assignments introduced here are:

<PREP> ← [MKADVPP <corresponding lexical item inJapanese>]

*ap ← [MKMKADJPP ""].

Example. "system for communication"

EFR: λx[(a*(communication))(λy[(((*ap(for))(y))(system))(x)])]

(step 1) Evaluating a CPSF assigned to a*(communication), we get a CPS structure of category $NP_{FUNCTION}$: [$NP_{FUNCTION}$ "通信"].

(step 2) A variable y is associated with a CPS structure: [NP "通信"].

(step 3) From a CPSF formula for *ap(for), we get a CPS structure:

[MKADJPP "のための"].

(step 4) From a CPSF formula for (*ap(for))(y), we get a CPS structure:

[ADJ [NP "通信"] [MKADJPP "のための"]],

using the first application rule introduced in this section.

(step 5) From a CPSF formula for ((*ap(for))(y))(system), we get a CPS structure:

[NOUN [ADJ [NP "通信"] [MKADJPP "のための"]]
[NOUN "システム"]],

using the second application rule introduced in this section.

Figure 5.10 illustrates the above process.

5.3.5.3 Relative Clauses

The form of an EFR expression for a relative clause is written:

which(λx[... x ...]).

A lambda form, which is of type <0,1> or a one-place predicate, corresponds to a relative clause, and a variable x is a place holder for a hole in the relative clause. An operator "which" changes the type from a one-place predicate to adjective or type <<0,1>,<0,1>>. This form as a whole is a functor to a head noun. A

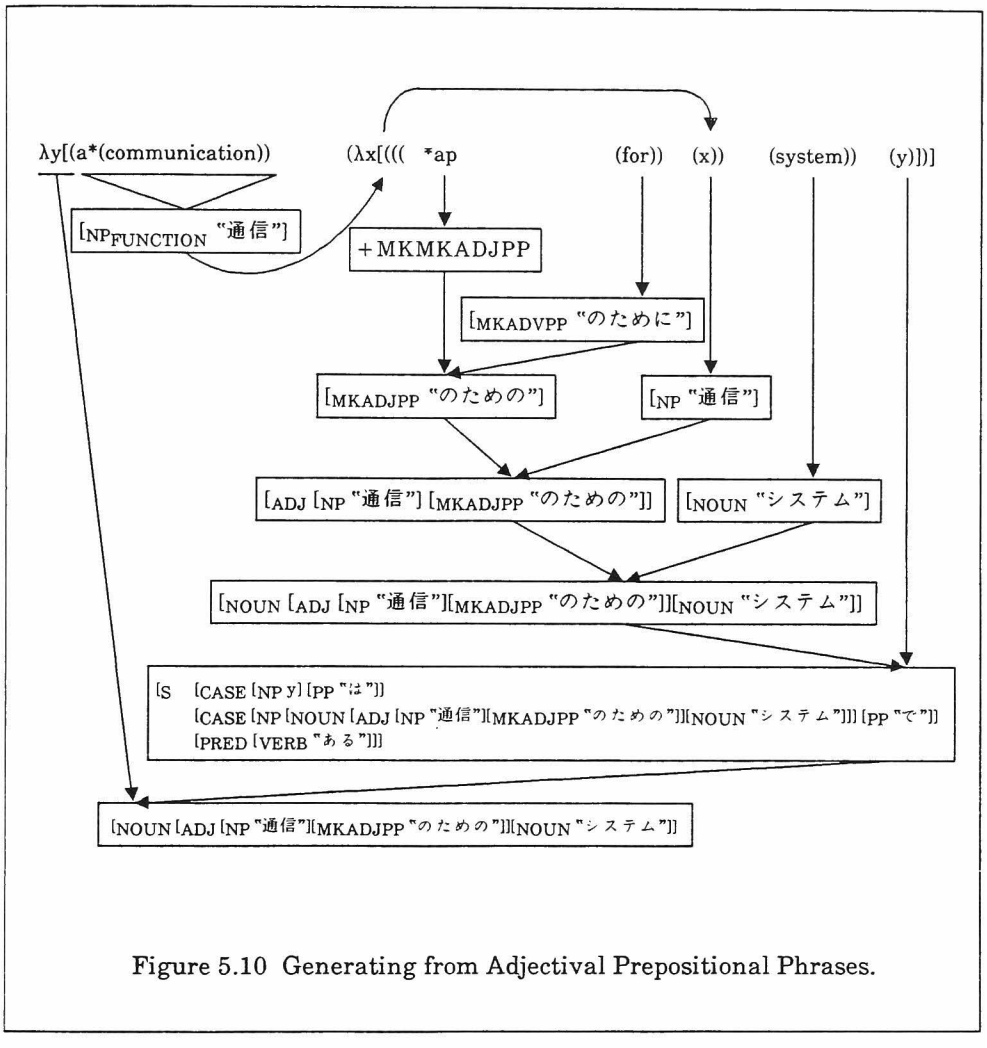


Figure 5.10 Generating from Adjectival Prepositional Phrases.

CPSF formula assigned to this form must be something which will attach a modifier phrase to a given CPSF function of category noun.

We will assign a CPSF function as follows to a lexical item "which":

$$\lambda p[\lambda q[\ \{NP_{FUNCTION}\ \ (+MKADJ(p(+MKHOLE(q))))(q)\}\ ]].$$

A variable p will be associated with a CPSF formula for a relative clause, and a variable q will be associated with a CPSF formula for a head noun. A function +MKADJ will generate a CPS structure of category adjective out of a CPS structure of category sentence. A function +MKHOLE will create a dummy noun

phrase from a given noun. This dummy noun phrase has the same feature complex as a given noun but it produces a null string. This dummy noun phrase is used as a place holder for a hole in a relative clause. Note that this lambda form is designed so as to transfer the conceptual information about the head noun to the place holder for a hole in the relative clause, enabling a word choice mechanism to work appropriately in relative clause construction.

For example, an EFR expression for a relative clause:

"(a) system which he develops"

is:

(which(λx[he(λy[develop(y,x)])]))(system).

By replacing each lexical item by an appropriate CPSF function, we get:

(λp[λq[{$_{NP_{FUNCTION}}$ (+MKADJ(p(+MKHOLE(q))))(q)}]]
(λx[[$_{NP_{FUNCTION}}$ "彼"](λy[develop*(y,x)])]))([$_{NOUN}$ "システム" with class=system]),

where,

develop* ≡ λxy[{class(y)=system → {$_{S}$ "xがyを開発する"};
else → {$_{S}$ "xがyを現像する"}}].

This form will evaluate to:

λq[{$_{NP_{FUNCTION}}$
(+MKADJ
((λx[develop*([$_{NP}$ "彼"],x)])(+MKHOLE(q))))
(q)}]
([$_{NOUN}$ "システム" with class=system])

⇒ {$_{NP_{FUNCTION}}$
(+MKADJ
((λx[develop*([$_{NP}$ "彼"],x)])
(+MKHOLE([$_{NOUN}$ "システム" with class=system])))
([$_{NOUN}$ "システム" with class=system])}

⇒ {$_{NP_{FUNCTION}}$
(+MKADJ

((λx[develop*([NP "彼"],x)])
([NP "" with class=system]))([NOUN "システム" with class=system])}

⇒ {NP FUNCTION
(+MKADJ
(develop*([NP "彼"],[NP "" with class=system]))
([NOUN "システム" with class=system])}

⇒ {NP FUNCTION
(+MKADJ
([S "彼が(を)開発する"]))
([NOUN "システム" with class=system])}

⇒ {NP FUNCTION
[ADJ [S "彼が(を)開発する"] [ADJPP "#RT"]]
([NOUN "システム" with class=system])}

⇒ [NP FUNCTION [NOUN [ADJ [S "彼が(を)開発する"] [ADJPP "#RT"]]
[NOUN "システム" with class=system]]]

If the input is:

"(a) film which he develops",

almost the same derivation will be carried out, and we will obtain a CPS formula:

[NP FUNCTION [NOUN [ADJ [S "彼が(を)現像する"] [ADJPP "#RT"]]
[NOUN "フィルム" with class=film]]].

Note that in an intermediate stage of evaluation, we have:

{NP FUNCTION
(+MKADJ
(develop*([NP "彼"],[NP "" with class=film]))
([NOUN "フィルム" with class=film])}.

Thus, the conceptual information of the head noun is transfered to the place holder for the hole.

5.3.5.4 Present and Past Participles

The form of an EFR expression associated with a present and past participle phrase is:

parti(<EFR expression for a verb phrase>).

As we observed in semantic analysis, present and past participles as noun modifiers can be paraphrased into a relative clause:

transmitted packet → packet which is transmitted
running machine → machine which is running.

Accordingly, we can assign to a functor "parti" the same CPSF function as that assigned to a reletive pronoun:

parti ← λp[λq[{$NP_{FUNCTION}$ (+MKADJ(p(+MKHOLE(q))))(q)}]].

5.3.5.5 Infinitives of adjective use

The form of an EFR expression associated with an infinitive in adjective use is:

inf-adj(<EFR expression for a verb phrase>).

The assignment to the operator "inf-adj" is:

inf-adj ← λp[{$_{ADJ}$ p(COMP*),[$_{ADJPP}$ "ための"]}].

5.3.6 Generating from Sentence Modifiers

5.3.6.1 Sentence Adverbs

Sentence adverbs are simply assigned a CPS structure of category ADV. For example,

yesterday ← [$_{ADV}$ "昨日"].

We introduce a new application rule to deal with the case where a CPS structure of category ADV is applied to a sentence:

[$_{ADV}$ x]([$_{S}$ y]) ⇒ [$_{S}$ [$_{ADV}$ x][$_{S}$ y]].

This rule specifies that if a sentence is modifed by a CPS structure of category ADV or sentence modifier, then the sentence modifier will be attached in front of the sentence.

5.3.6.2 Adverbial Prepositional Phrases

The form of EFR expression assigned to an adverbial prepositional phrase is slightly different from that for adjectival prepositional phrases. The general form of EFR expression for an adverbial prepositional phrase is:

$$\lambda x[\lambda s[<NP>(\lambda y[(<PREP>(y))(s)])]].$$

We introduce a new application rule for convenience:

$$[_{MKADVPP}\ x]([_{NP}\ y]) \Rightarrow [_{ADV}\ [_{NP}\ y][_{MKADVPP}\ x]].$$

This rule says that if a CPS structure of category MKADVPP or adverb-making postposition is applied to a noun phrase, the result is an adverbial phrase with the noun phrase followed by the adverb-making postposition.

CPSF assignment to a preposition is the same as that for adjectival prepositional phrases. An example is shown below:

Input sentence: "the packet arrives at the destination without error"

EFR expression: (the(packet))
(λx[(a*(error))
(λy[((without(y))((the(destination))(λz[arrive-at(x,z)])))])])

CPSF assignment here includes:

without ← [MKADVPP "なしに"].

Evaluation goes on as follows:

(step 1) A variable x is bound to [NP "そのパケット"].

(step 2) A variable y is bound to [NP "(ある)エラー"].

(step 3) A variable z is bound to [NP "(その)目的地"].

(step 4) Evaluating a CPSF formula for without(y), we get:

[ADV [NP "(ある)エラー"][MKADVPP "なしに"]].

(step 5) Evaluating a CPSF formula for:

(the(destination))(λz[arrive-at(x,z)]),

we get:

[S "そのパケットが(その)目的地に到着する"].

(step 6) Evaluating a CPSF formula for:

((without(y))((the(destination))(λz[arrive-at(x,z)]))

we get:

[S [ADV [NP "(ある)エラー"][MKADVPP "なしに"]]
[S "そのパケットが(その)目的地に到着する"]].

5.3.6.3 Infinitives in Adverbial Use

The form of EFR expression for infinitives in adverbial use is:

inf-adv(<EFR expression for a verb phrase>).

The assignment we make in this case is:

inf-adv ← λv[{ADV v(COMP*),[MKADVPP "のために"]].

When this CPSF function is applied to a CPSF function for a verb phrase, it will produce a sentence modifier by first applying a CPSF function for the verb phrase to a dummy noun phrase COMP* then attaching an adverb making postposition to the resulting sentence.

5.3.7 Noun Clauses

The EFR expression associated with noun clauses consists of an EFR expression corresponding to a complimentizer and an EFR expression for embedded sentences. The form of EFR expression is:

<EFR expression for a complimentizer>(<EFR expression for an embedded sentence>).

In this case, we simply make assignments such as:

that ← [MKNP$_{FUNCTION}$PP "こと"]

whether ← [MKNP$_{FUNCTION}$PP "かどうか"].

We introduce a new application rule to deal with cases where these CPSF fuctions are applied to a CPS structure of category S:

[MKNP$_{FUNCTION}$PP x]([S y]) ⇒ [NP$_{FUNCTION}$PP [S y][MKNPPP x]].

We give similar treatment to infinitives used as nouns. The form of EFR expression for an infinitive used as nouns is:

inf-n(<EFR expression for averb phrase>).

Our assignment is:

inf-n ← λv[[MKNP$_{\text{FUNCTION}}$PP "こと"](v(COMP*))].

5.3.8 Dealing with Mood

5.3.8.1 Interrogatives

In dealing with interrogative sentences, we first create an indirect question and then we transform it into a direct question. This is related to our semantic treatment of interrogatives; we paraphrase a question such as:

Does the machine understand natural language?

or, What do you have?

into a declarative sentence as:

I ask you whether the machine understand natural language,
or I ask you what you have.

The form of EFR expression for yes/no question is:

#ques(whether(<an EFR expression for a declarative sentence>)).

The form of an EFR expression for WH-question, on the other hand, differs from case to case. Lexical items "when" and "whether" are analyzed as a functor to EFR expression of type 0, while constituents "who", "what", "which + <NOUN>", and "how + <ADJ> + <NOUN>" are analyzed as a functor to EFR expressions of type <0,1>. Figure 5.11 illustrates examples.

As described in the treatment of noun clauses, we assign to an operator "whether" a complimentizer:

[NP$_{\text{FUNCTION}}$PP "かどうか"].

On the other hand, we substitute into <WH-operators> different CPSF functions as follows:

(1) "When did you come back?" ⇒ #ques(when(you(did(come-back))))

(2) "Where did you meet him?" ⇒ #ques(where(you(λx[he(λy[did(meet(x,y))])])))

(3) "Who wrote the program?" ⇒ #ques(who(λx[(the(program))
(λy[did(write(x,y))])]))

(4) "What did you buy?" ⇒ #ques(what(λy[you(λx[did(buy(x,y))])]))

(5) "Which car did you buy?" ⇒ #ques((which(car))(λy[you(λx[did(buy(x,y))])]))

(6) "How many books do you have?" ⇒ #ques(((how(many))(*pl(book)))
(λy[you(λx[have(x,y)])))

Figure 5.11 Example of WH-Questions and EFR Expressions for Them.

when ← λs[[$MKNP_{FUNCTION}$PP "か"]({S [QADV "いつ"],s})]

where ← λs[[$MKNP_{FUNCTION}$PP "か"]({S [QADV "どこで"],s})]

who ← λp[[$MKNP_{FUNCTION}$PP "か"](p([QNP "誰"]))]

what ← λp[[$MKNP_{FUNCTION}$PP "か"](p([QNP "何"]))]

which ← λn[λp[[$MKNP_{FUNCTION}$PP "か"](p([QDET "どの"](n)))]]

how ← λa[λn[λp[[$MKNP_{FUNCTION}$PP "か"](p([MKQDET "どれだけ"](a))(n)))]]].

In order to carry out the above computation, we introduce the following application rules:

[QDET x]([NOUN y]) ⇒ [QNP [QDET x][NOUN y]]

[MKQDET x]([ADJ y]) ⇒ [QDET [MKQDET x] [ADJ y]]

In order to transform those indirect questions into direct questions, we introduce a transformation as shown in figure 5.12.

Figure 5.13 shows how CPS structures are generated from our initial examples.

5.3.8.2 Imperatives

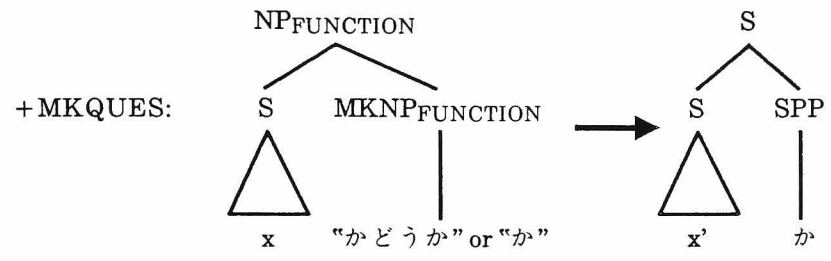


Figure 5.12 Transforming Indirect Questions into Direct Questions.

Our treatment of imperatives is similar to that for interrogatives. In translating imperatives into EFR, we make a paraphrase as follows:

Read this instruction. → I order you to read this instruction.

And the form of EFR expresion assigned to imperatives is:

#imp(inf-n(<an EFR expression for a verb phrase>)).

Evaluating a CPSF formula assigned to a subexpression:

inf-n(<an EFR expression for a verb phrase>),

we obtain a CPS structure of category $NP_{FUNCTION}$:

$[NP_{FUNCTION}$ [S ...][MKNPPP "こと"]].

In order to transform this structure into an imperative sentence, we introduce a transformation as follows:

+MKIMP: $[NP_{FUNCTION}$ [S x][MKNPPP "こと"]] → [S [S x][MKSPP "よ"]].

In CPSF assignment, we simply replace a lexical item "#IMP" by this transformation operator. Thus,

#IMP ← +MKIMP.

Figure 5.14 illustrates an example.

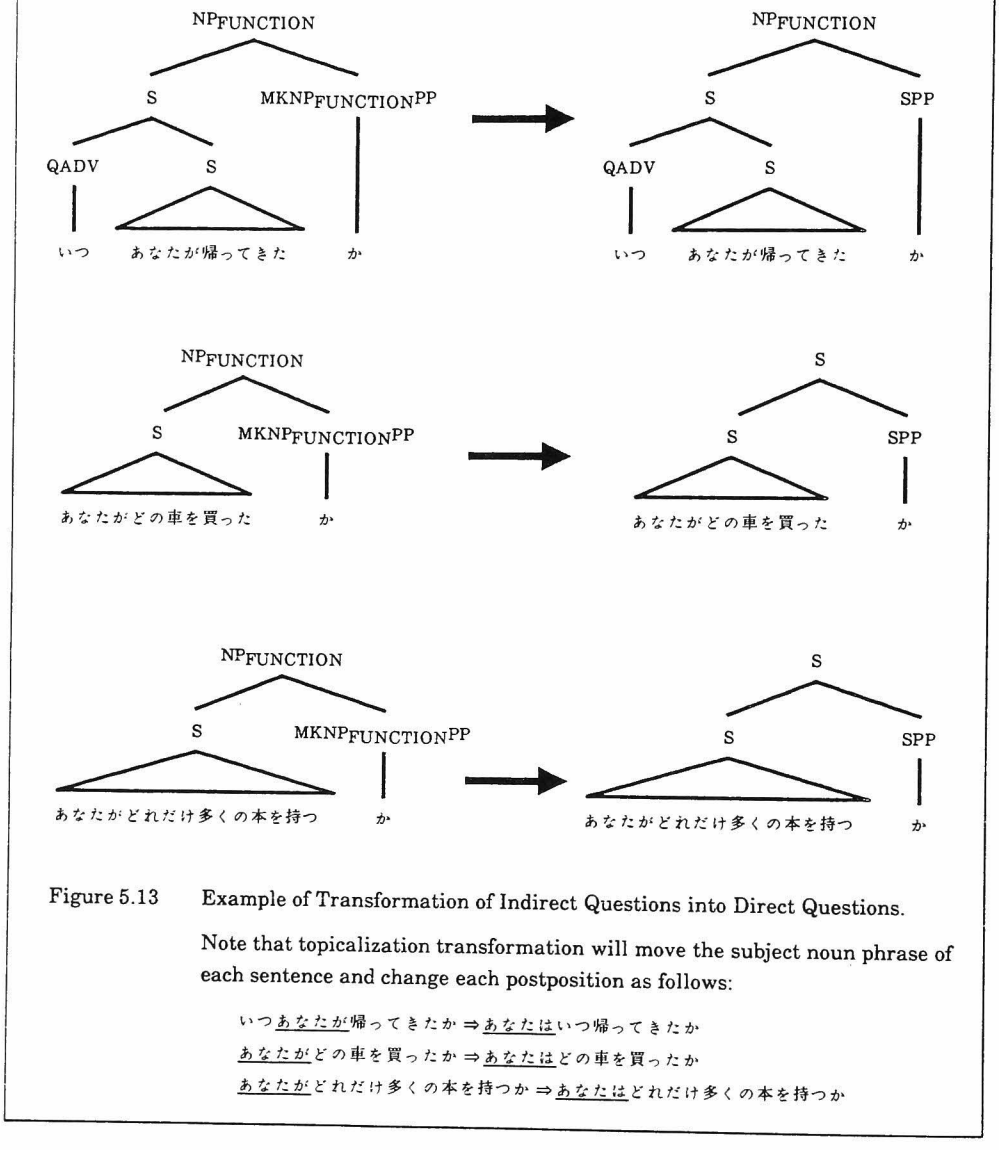


Figure 5.13 Example of Transformation of Indirect Questions into Direct Questions.

Note that topicalization transformation will move the subject noun phrase of each sentence and change each postposition as follows:

いつあなたが帰ってきたか ⇒あなたはいつ帰ってきたか
あなたがどの車を買ったか ⇒あなたはどの車を買ったか
あなたがどれだけ多くの本を持つか ⇒あなたはどれだけ多くの本を持つか

5.4 Application of Heuristic Rules

The CPS structure of the target language obtained from evaluation of CPSF function may contain a number of lengthy, inadequate, or errorneous expressions. Such substructures are created for several reasons:

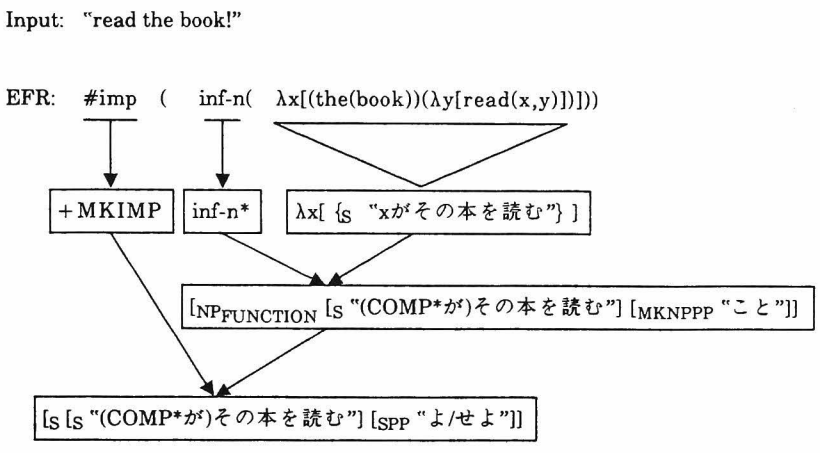


Figure 5.14 Generating CPS Structure from Imperative.
where, inf-n* ≡ λp[{NPFUNCTION p(COMP*),[MKNPPP "こと"]}].

(a) Discourse structure was not taken into account.

(b) Global aspects were not considered.

(c) Analysis is not sufficiently deep.

(d) Ambiguities are newly introduced in the generation stage.

We use a set of tree-tree rewriting rules to automatically detect inadequate substructures and to replace them by more adequate expressions. Each rewriting rule consists of a structural description of the structure to be modified, and a description of how it is modified.

5.5 Morphological Synthesis

In morphological synthesis, we make inflectional processing for each predicative word (YOUGEN). Table 5.2 shows the inflectional suffix for predicative words of each part of speech. Which inflectional suffix is selected depends on the succeeding word. The rule of inflection involves a number of exceptional cases and is not very simple.

Table 5.2 Inflectional Suffics of Japanese.

Inflectional Suffix / Type of Inflection	1	2	3	4	5	6	7	8	9	10
GODAN	A	I	U	U	E	E	O			
-W	WA	I	U	U	E	E	WO			
+VP2	Ø	Ø	RU	RU	RE	RO	YO			
SAHEN	SI	SI	SURU	SURU	SURE	SEYO	SI	SE	SIRO	SA
KAHEN	KO	KI	KURU	KURU	KURE	KOI				
KYO	KARO	KU	I	I	KERE	Ø	KAT			
KDO	DARO	NI	DA	NA	NARA	Ø	DAT	DE		
TA	RO	Ø	Ø	Ø	RA	Ø				

where, Inflectional Suffix:

1 (MIZEN) : Imperfect
2 (RENYOU) : Adverbial
3 (SHUUSHI) : Stop
4 (RENTAI) : Adjectival
5 (KATEI) : Subjunctive
6 (MEIREI) : Imperative
7 - 10 : for exceptional cases

Type of Inflection:

GODAN, -W, +VP2, SAHEN, KAHEN : verbs
KYO, KDO : adjectives
TA : auxiliary verbs

In doing inflectional processing, we use a triple called a *terminal string frame* to represent a property of a terminal sequence. A terminal string frame is represented:

(<RomanCharacterString> <KANJICodeString> <AttributeValueList>).

AttributeValueList contains information about morphological properties of the terminal sequence. We use the following set of attributes:

CAT: syntactic category of the governing word.

-KATUYO: instruction of inflection to the preceding word. For example,

#NAI: ((...)(...)(..(-KATUYO.1)...)

signals the preceding word that the first (MIZEN or imperfect) form of inflectional suffix should be taken.

KTYPE : inflection type of governer.

- : splitted sound: K, B, or M.

IF- : an exception-handling procedure.

The morphological dictionary consists of lexical item and terminal string frame pairs.

General Algorithm for Processing Inflections

Figure 5.15 gives a perspective of how a terminal string is generated from a phrase structure (a CPS structure, to be more precise).

A Procedure #LINEAR creates a terminal frame from a given phrase structure α. #LINEAR calls a subroutine #CONCAT(p,q) to merge two terminal string frames p and q into one. The resulting terminal string frame mainly inherits the properties of q. Figure 5.16 shows the internal processing of #LINEAR and #CONCAT.

Exception Handling

Inflectional processing includes a number of exceptional cases:

(a) sometimes, the subsequent word should be changed into a voiced sound. For example, TESIMAU/DESIMAU, TA/DA, TE/DE.

(b) sometimes, the indication of inflection by a preeding word is not simple, e.g., ONBIN.

To make it easy to handle some exceptional cases, we need to split the word stem of some verbs into two parts. For example, the inflectional type of "TOBU" (fly) is GODAN; its inflectional postposition changes as: TOB-A(-NAI), TOB-I(-MASU), TOB-U, TOB-U(-TOKI), TOB-E(-BA), TOB-E. But it takes the ONBIN form as: TO-N-DA. In the morphological dictionary, the stem of "TOBU" is written as "TO" and the remaining sound "B" is put into the - attribute. Thus,

TOB: ((TO) (飛) ((KTYPE . GODAN) (- . B))).

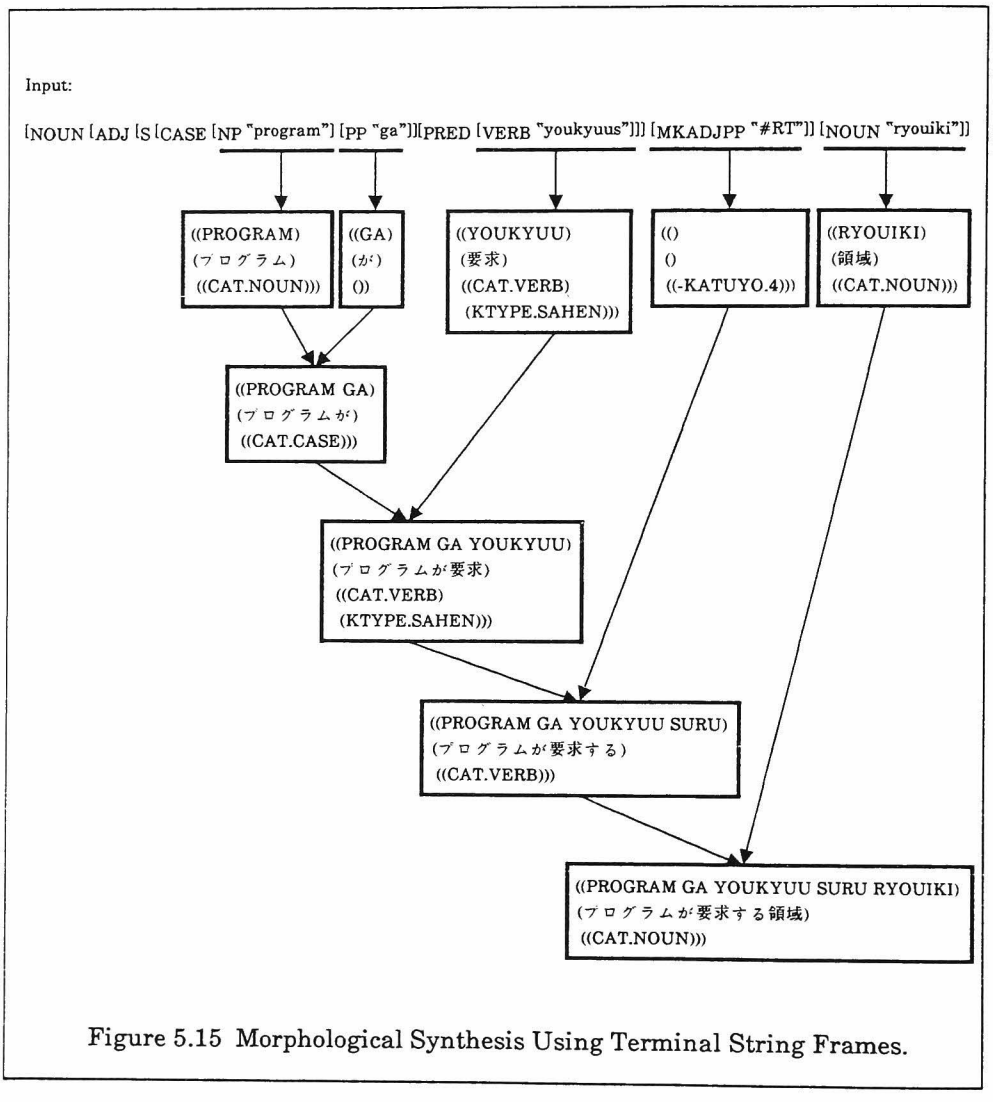


Figure 5.15 Morphological Synthesis Using Terminal String Frames.

Dealing with Voiced Sound. We distinguish a voiced sound version of a lexical item (e.g., DESIMAU) from the original one (e.g., TESIMAU). We use the exception handling feature to replace the lexical item:

(IF- (<condition>$_i$ <new lexical item>$_i$) ...),

If <condition>$_i$ holds, the current lexical item will be replaced by <new lexical item>$_i$.

#LINEAR(x): (x: node of phrase structure)

if x is a terminal node, **then** consult morphological dictionary;
else (when x is a nonterminal node)
connect immediate descendants from right to left using a subroutine #CONCAT. Thus,

#CONCAT(#LINEAR(B_1), ... ,#CONCAT(#LINEAR(B_{n-1}),#LINEAR(B_n)) ...),
where B_1, ..., B_n are immediate descendants of x.

#CONCAT(p,q): (p and q are terminal string frames)
if an exceptional processing for q is defined, **then** execute the procedure;
if inflectional processing is necessary for p **then** do it;
let the new terminal string be an appended string of:
p's terminal string,
inflectional postfix of p (if any), and
q's terminal string;
let the attribute of the new terminal string frame be the same as q's.

Figure 5.16 General Procedures for Inflectional Processing.

For example, we can assign a terminal string frame to a lexical item TESIMAU as follows:

#TESIMAU:(... ((IF- ((KTYPE GODAN) (- (*OR B G M))) #DESIMAU)) ...)

According to this assignment, if the inflectional type of the preceding word is GODAN and if its last sound is B, G, or M, then #TESIMAU is replaced by #DESIMAU. Thus,

TOB + #TESIMAU → TO-N-DESIMAU
"fly"

A sound "N" is inserted through ONBIN processing as described below.

ONBIN Processing. ONBIN processing is needed if a class of verbs of inflectional type GODAN follows a word whose sound starts with "T" or "D" and which requests the verb to take the second (REN-YOU) inflectional suffix.

We extend the initial form for signaling inflectional suffixes as follows to handle exceptional cases:

(-KATUYO . (n ... (ktype$_i$. n$_i$) ...),

if the inflectional type of the preceding word is ktype$_i$, then its inflectional suffix should be as indicated by n$_i$, otherwise the inflectional suffix is n-th one.

Using this extended form, ONBIN is indicated as follows:

DA: ((...) (...) (... (-KATUYO.(2 *GODAN . ONBIN) (-W ONBIN) ...))).

Then the inflectional suffix is determined as follows:

Terminal string frame of the preceeding word: (... (... (- . x) (KTYPE . w) ...))

Terminal string frame of the subsequent word: (... (... (-KATUYO . ONBIN) ...))

Procedure to determine the inflectional suffix:

if y = GODAN
  then **if** x ∈ {A,T,R,W} ∨ word-stem = I **then** c ← T;
    else if x ∈ {N,M,B} **then** c ← N;
    else if x ∈ {K,G} **then** c ← I;
    else c ← ∅;
else if y = -W **then** c ← T;
else c ← ∅.

Resulting string:

<the terminal string of the preceding word>-c-<the terminal string of the subsequent word>

Examples. Using terminal strings:

HASIR: ((HASI) (走) ((KTYPE . GODAN) (- . R))),

TIJIM: ((TIJI) (縮) ((KTYPE . GODAN) (- . M))),

TOK: ((TO) (解) ((KTYPE . GODAN) (- . K))),

IW: ((I) (言) ((KTYPE . -W))),

we get:

HASIR + TA → HASI-T-TA (走った)

TIJIM + TA → TIJI-N-DA (縮んだ)

TOK + TA → TO-I-TA (解いた)

IW + TA → I-T-TA (言った).

There are some other cases in which exceptional handling is needed. In such cases, the exception-handling procedure in -KATUYO feature is also utilized. For example, we write:

#RERU: ((RE) (れ) ((-KATUYO . (1 (SAHEN . 10))) (- . R) ...))

to deal with the cases as follows:

KOWAS + #RERU → KOWAS-A-RERU (壊される)
(KTYPE=GODAN)

HAKAIS + #RERU HAKAI-SA-RERU (破壊される)
(KTYPE=SAHEN)

6. Experiments

Generally, there are four levels of research in machine translation.

- purely theoretical work

- building a demonstratable prototype

- building a comprehensive system

- constructing a commercial system.

We began with the first level and have completed the second level. We have built an experimental system and carried out experiments for a number of sample sentences taken from scientific literature.

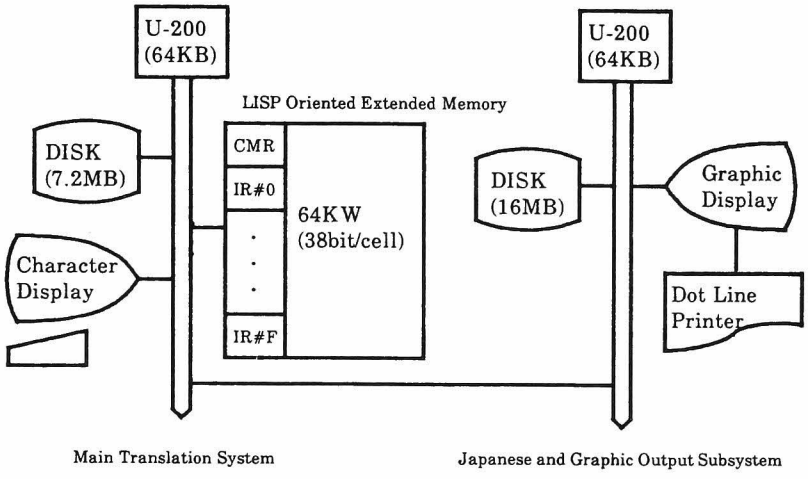
Experimental System

The experimental system was built on a LISP 1.7 system (42Kcell) [Doshita 78a], which was installed on a minicomputer. We extended the basic facility of the LISP system to make it possible to carry out a nontrivial number of experiments. Facilities added to the original LISP system include a dictionary subsystem, KANJI I/O and graphic output. The experimental system consists of about 6000 lines in LISP. Currently, about 160 analysis rules are defined and about 800 items are registered in each dictionary. As it consists of a number of independent modules, the programming effort required to build the system was relatively small. Figure 6.1 shows the software and hardware organization.

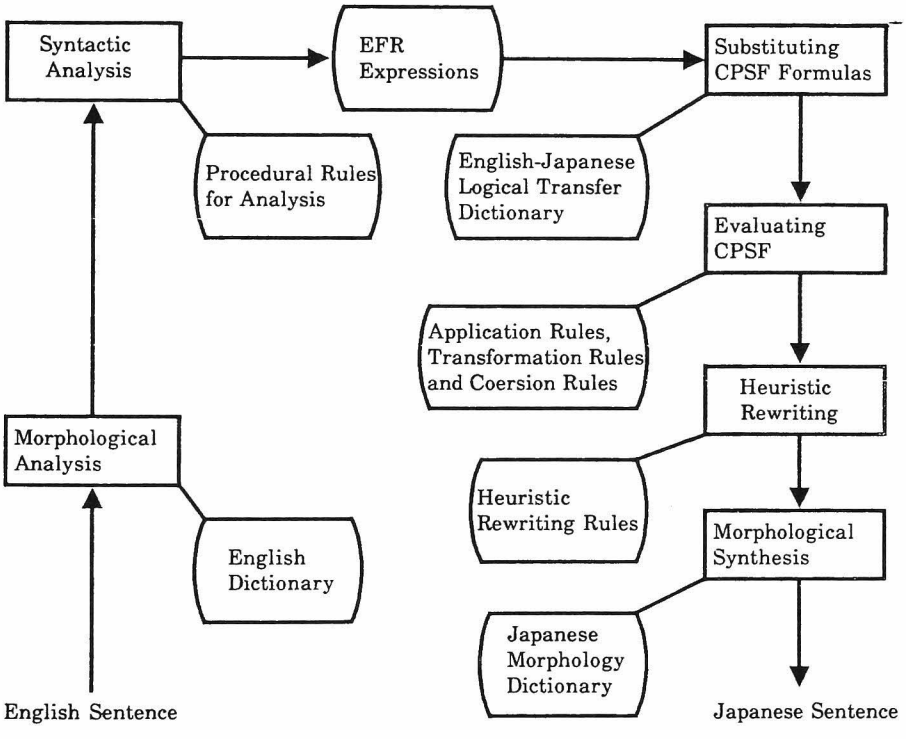
Actual grammar rules and dictionary entries are based on a finer classification than described in this thesis; in particular verb patterns used there is based on Hornby's classification[Hornby 74,75].

Human Assistance

The experimental system requires some amount of human assistance.



(a) Hardware Organization



(b) Software Organization

Figure 6.1 Hardware and Software Organization of the Experimental English-Japanese Machine Translation System.

First, the system compelled us to do some pre-editing, although this was restricted to rather minor issues, as follows:

"assembly language" → "the assembly language"

"object program" → "an object program"

"16- or 32-bit" → "16-bit or 32-bit"

Second, the user has to resolve ambiguities. Ordinary sentences consisting of 20-30 words were analyzed in many ways. For example, a sentence:

> "The assembly language provides us a means for writing programs without having to be concerned about actual memory addresses or machine instruction formats."

was analyzed in 78 ways (figure 6.2).

Among the outputs from the system, the correct analysis was 43-rd one. Interactive diagnosis facility is useful to some degree in decreasing the number of interactions required to obtain the correct analysis. In the above case, for example, the correct analysis was obtained after two cycles of diagnosis and rejection. Note that, however, this discussion does not to imply that the interactive diagnosis is sufficient for solving ambiguities but simply means that *if* the knowledge encoded in the grammar rules is limited *then* the interactive diagnosis is useful.

Finally, there is implicit human assistance for dealing with unknown words or unknown phrases. When the results of translation are completely inadequate, the designer must improve the grammar rules.

Experimental Results

The system can translate sample sentences successfully. Part of the results are shown in the appendix.

The results, however, contain a number of errors. Generally, the correctness of the output depends on the adequacy of the analysis. But some errors originate in the generation phase. Figure 6.3 shows typical errors.

However, all of these difficulties derive from the poorness of the linguistic data incorporated into the system and do not imply the inadequacy of the framework.

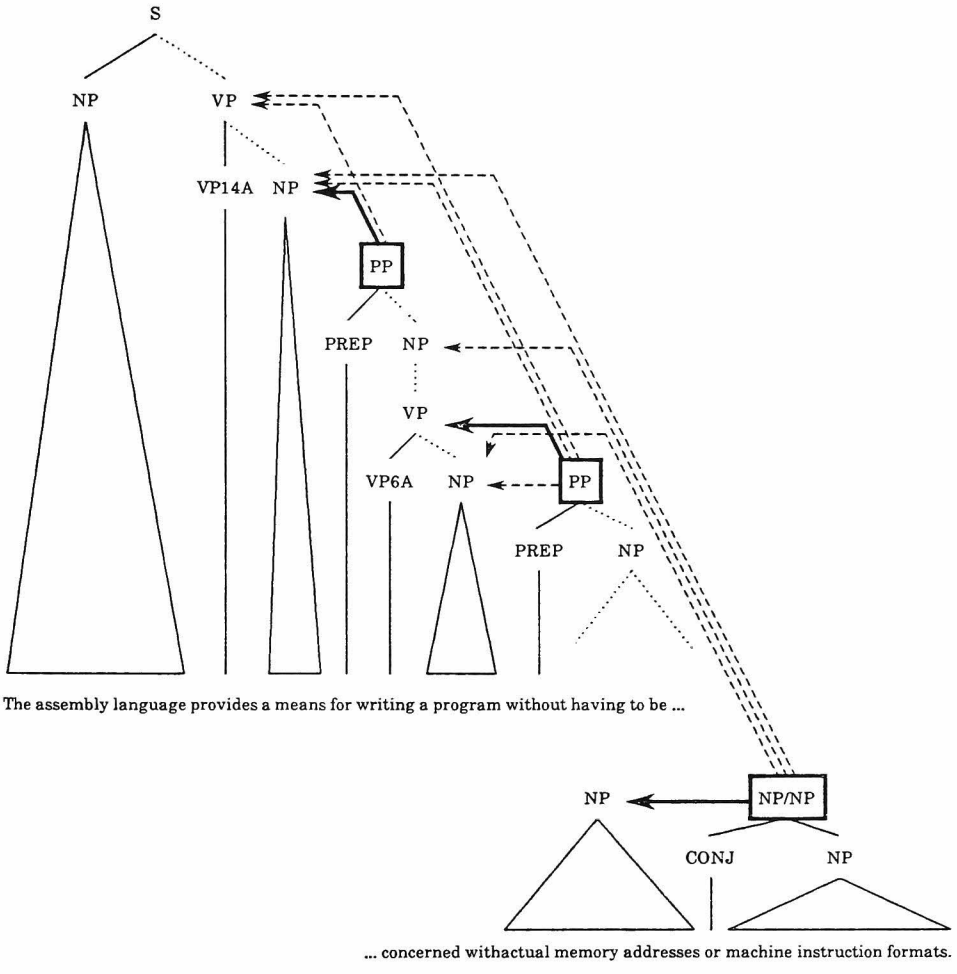


Figure 6.2 Source of Ambiguities Arising from the Analysis of a Sentence:

"The assembly language provides a means for writing a program without having to be concerned with actual memory addresses or machine instruction formats."

Note that there are more ambiguities arising from the internal structure of complex nouns: "actual memory addresses" and "machine instruction formats"

(a) Ethernet is a branching broadcast communication system for carrying digital data packets among locally distributed computing stations.

ETHERNETはデジタルデータパケットを局所的に分散される計算ステーションの間に伝搬するための分岐状の放送通信システムである。

(b) The packet transport mechanism provided by Ethernet has been used to build systems which can be viewed as either local computer networks or loosely coupled multiprocessors.

局所的な計算機ネットワークまたは疎に結合されるマルチプロセッサとみなされることができるシステムを構築するためにETHERNETによって提供されるパケット輸送機構は用いられてしまう。

(c) An Ethernet's shared communication facility, its ether, is a passive broadcast medium with no central control.

ETHERNETの共有された通信設備(そのETHER)は中央の制御を持たない受動性の放送媒体である。

(d) Coordination of access to the ether for packet broadcasts is distributed among the contending transmitting stations using controlled statistical arbitration.

パケット放送のためのそのETHERへのアクセスの調整は制御される統計的調停を用いて競合している送信側のステーションに分散される。

(e) Switching of packets to their destinations on the ether is distributed among the receiving stations using packet address recognition.

そのETHERの上の目的地へのパケットの交換はパケット番地認識を用いて受信側のステーションの間に分散される。

(f) Design principles and implementation are described based on experience with an operating ethernet of 100 nodes along a kilometer of coaxial cable.

設計原則と実働化は1Kmのコアキシャルケーブルに沿った100のノードの稼動中のETHERNETについての経験に基づいて述べられる。

(g) A model for estimating performance under heavy loads and a packet protocol for error-controlled communications are included for completeness.

重い負荷の下での性能を評価するためのモデルと誤り制御された通信のためのパケットプロトコルは 完全のために含まれる。

Input text is cited from: Metcalfe,R.M. and Boggs,D.R., Ethernet: Distributed Packet Switching for Local Computer Networks, CSL-75-7, Xerox Palo Alto Research Center, (1980).

Figure 6.3 Typical Errors in the Experimental Machine Tranlation System.

Both translation errors and other improper translations are underlined.

Rather, what is important is the fact that we were able to easily implement the experimental system with a small amount of programming effort.

In the next chapter we further discuss the problems with the current system and possible revisions.

7. Discussion

In this chapter, we discuss the framework we proposed and make comparison with other work.

Application of Montague Grammar to Natural Language Processing

A number of computer scientists have attempted to apply Montague grammar to a natural language processing including machine translation. Early researchers mainly attempted to construct a program which could simulate the framework of Montague grammar. For example, Friedman's program calculates a truth value of a given sentence based on the given model[Friedman 78a,b]. The objective of the system was to teach students the concept introduced by Montague. There have been other attempts at constructing a parser and evaluator based on Montague grammar[Indurkhya 81], [Matsumoto 81]. Itensional logic has been used not only as a formal tool for natural language analysis[Hobbs 78], [Friedman 79, 80, 81], [Janssen 77, 80a,c, 81], [Landsbergen 81], but also as a basis for mathematical models of the semantics of programming language or data bases[Janssen 78, 80b], [Yonezaki 80a,b]. This reflects the fact that logic has been widely used for natural language analysis[Scha 80],[Habel 80],[Charniak 81],[Gawron 82],[Rosenschein 82] and for formalizing common sense reasoning[Moore 77].

However, as is discussed in chapter 3, the framework of Montague grammar cannot be directly applied to natural language understanding, since it is a "generative" framework. *Given* a perfect description about the real world, Montague grammar can provide a procedure to compute the truth value. Apparently, however, we do not have such complete knowledge in advance; rather the world knowledge must be something acquired through reading or hearing sentences. What is important in natural language understanding seems to consist of such mental processes in building a consistent world model from input sentences. Montague grammar tells us nothing about how to do such processing. Indurkhya (81) uses meaning postulates to simplify intensional logic forms, but such an approach is very limited since intensional logic is not a sound framework

for making inferences. Some other researchers simply utilize Montague's framework as a mechanical translator into first order logic. First-order logic is used both as a device for semantic representation and as a device for making inferences.

We used a semantic network formalism. Semantic network formalism stresses the aspect of association. It has been used for a number of purposes: as a model of human memory[Quillian 68], as a formalism for storing problem-solving knowledge[Fikes 77], [Tsujii 78], or as a semantic representation[Simmons 73],[Yoshida 79]. We use semantic networks for all of these three purposes. We use semantic networks as a memory structure for representing new information obtained from analyzing input sentences. Semantic networks created as a result of our semantic interpretation procedure may contain uninterpreted issues, which will be further analyzed in a domain specific context. In this sense, our semantic network can be viewed as a sort of semantic representation. To each node of our semantic networks, a procedure for making inference can be attached. Such procedures can be invoked using inheritance. In this sense, our semantic networks can be viewed as a problem solver for semantic interpretation.

Semantic Network as Semantic Representation

Our semantic network gives a partial solution to the problems Woods (75) pointed out. Our formalism involves intensional representation, quantification, structures for relative clauses, and representation for definite and indefinite noun phrases, some of which will be simplified in the accommodation process, for example by resolving reference. We do not use function nodes such as those used in Tsujii (78). Information represented by function nodes can be replaced by a combination of a relation node and a definite node. If there is a need to calculate the value of a functional representation, it can be done by a procedure attached to relation nodes.

As to the role of the semantic network in natural language semantics, we treat it as a device for representing the results of semantic interpretation. In the process of what we call semantic interpretation, we make inference to accommodate new information into the knowledge base. For example, definite nodes are replaced by their referents. Note, however, that we cannot do this sort of processing completely, since natural language may contain phrases which cannot be simplified in this way. For example, we cannot replace the definite noun phrase "the number" by its referent in a sentence:

he does not know the number of the safe.

since a sentence resulting from the replacement:

he does not know 1123,

makes different sense. But basically what we do in semantic interpretation is to evaluate new information in terms of old information. In this sense, our system does more than those systems which simply translate natural language sentences into equivalent logical formulas.

In formal semantics, the meaning of a sentence can be understood as a function which gives the denotation once a discourse structure is fixed. In this sense, the meaning itself of a sentence:

I told you to come here tomorrow,

is not anything affected by facters such as who is the speaker, who is the hearer, where and when the sentence is uttered, etc. Instead, the meaning of the above sentence is a function which takes those facters as parameters and which will output the value if they are given. Computationally, however, it does not seem to be important to caluculate the definition of such a function from the input sentence. What we did with semantic networks was to represent the interpreted meaning of the sentence. Instead of calculating the truth value, we attempted to give a detailed description of the process of how semantic networks are created and put together by interpreting a sentence. The meaning of the input sentence in the sense of formal semantics can be thought of as a pair of the formal representation obtained from syntactic-semantic analysis and a procedure for semantic interpretation. This pair will take a discourse structure as a parameter and will create a semantic network. From a computational point of view, however, such a discussion does not make much sense. What is important is to discuss the process of semantic interpretation to detail or to investigate the characteristics of knowledge and inferences needed for semantic interpretation.

Semantic Network as a Problem Solver

Our semantic network is used as a problem solver for interpreting natural language. Problem solving facilities are attached procedures and the notion of inheritance. This formalism reflects an object-oriented view of problem solving knowledge. The range of problems solved with this framework is limited to those closely related to natural language semantics, so it does not necessarily have a

problem solving ability for a specific task domain. Such a task-oriented problem solver should be linked to the general problem solver as an external system. For example, in a question-answering environment, our semantic network system need not solve differential equations nor make inferences like those made in reading suspense or detective stories. A procedure for inferring the murderer is thought of as an external one.

Semantic Network as a Memory Model

Semantic networks store information in an object-centered manner. Links are used to make it possible to retrieve items by association. In this respect, we mainly use ever-developed ideas and have made little contribution. Although we introduced a device for representing intensional objects, for example, we did not describe how they are utilized. This problem should be answered in the future.

What is Claimed

In summary, what we want to claim for the semantic network model is:

That it serves as a both rigorous and powerful framework for semantic representation; it is closely related to logical representation but it is more feasible as a computational device for making inference or retrieving information.

That it serves as a problem-solving facility for organizing knowledge for understanding natural language utterances; object-centered representation and use of hierarchy provide a moderate way for organizing knowledge.

That the procedure for obtaining semantic network representaitons is defined as a stepwise process; semantic networks are created as a result of step-by-step application of functions.

What does the semantic network represent?

There still remains the problem of what the semantic network is. Computationally, this question can be answered operationally, by makig a virtual machine (actually a pair of a real machine and a program running on the machine) which can execute the accommodatioan and retrieval of semantic networks. As to the theretical aspects of semantic networks, researchers are attempting to give formal treatments. Most of the work has attempted to make associtions between semantic networks and first-order logic[Schurbert 76], [Allen 82]. The semantics of our

semantic network may be seen in this way, but this remains as a problem for the future.

Application of Montague Grammar to Machine Translation

As to the application of Montague grammar to machine translation, there are two systems whose reports are published. But those methods used there are quite different from ours.

The Rosetta System. Landsbergen (82) attempts to use the notion of *logically isomorphic Montague grammar* to give a basis for a multilingual machine translation system. This is based on the assumptions that:

> every language expression is generated by choosing an appropriate sequence of derivation rules or transformation rules,
>
> for any one derivation rule or transformation rule of one language, we can find an appropriate sequence of derivation rules or transformation rules in another language which adds the same meaning to the original expression.

Rosetta is characterized by the use of what they call *a logical derivation tree*, which stands for the deep derivation process. Figure 7.1 illustrates the translation process by Rosetta. The advantage of Rosetta is that the same grammar can be used for both analysis and generation. This claim is fruitful for multilingual translation. The success of Rosetta depends on whether or not they can actually define logically isomorphic grammars.

The problem with Rosetta is that their framework does not take semantic issues into consideration, or if it does, the treatment of semantic information is not explicit. It is obvious that to obtain high quality translation, semantic information is necessary to resolve ambiguities, to make word choice or to make semantic-oriented paraphrases. Accordingly, the framework of machine translation should have a component for dealing with semantic information. In this respect, the current framework of Rosetta seems to be inappropriate, since it does not allow semantic information to be incorporated into the system in an explicit form.

SALAT. Another example of a machine translation system based on Montague grammar is SALAT[Hauenschild 79]. SALAT was designed to be of primarily theoretical interest. Currently, rules are written for German-French translation. SALAT uses what they call ε-λ notation as a pivot language. For example, they assign to a sentence:

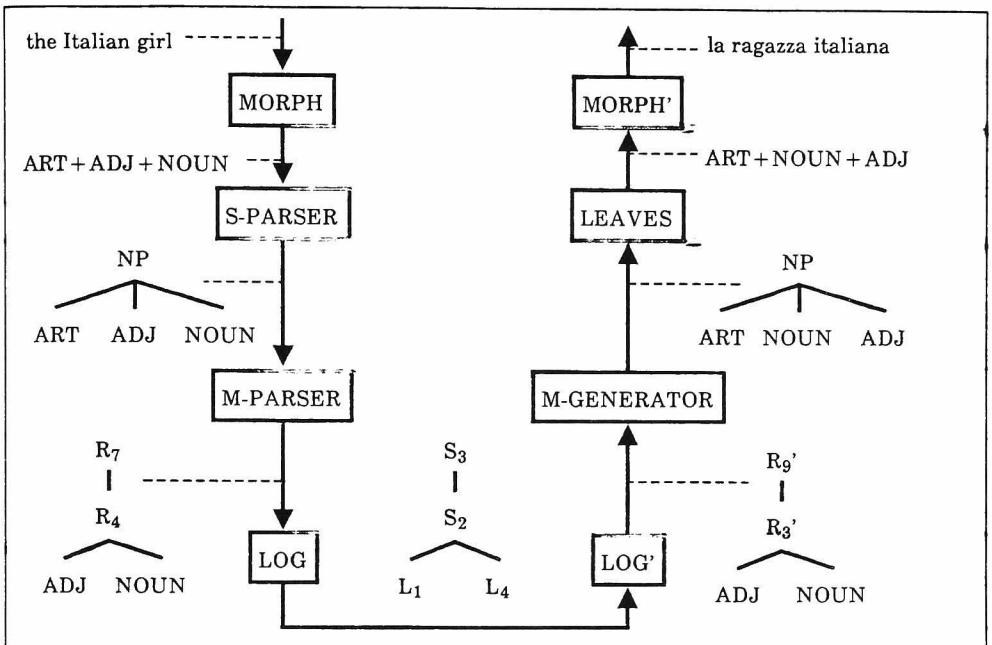


Figure 7.1 Translation Process Used by Rosetta.
(cited from Landsbergen (82), reproduced.)

Der Hund, der einen Harren hat, belt,
(the dog which has a master barks)

a "deep" structure:

$BELL $DEF x($HUND x & $INDEF y($HERR y & $HAB y x)).

Actually, ε-λ notation can be viewed as a sort of logical representation. SALAT makes deductions with this sort of notations to resolve ambiguities or to choose appropriate words (or phrases) in the target language. For example, in a German-French translation, the sense of a word "Gebiet" should be distinguished at least in two ways: region and domain. If "Gebiet" is used to mean a geographic concept, it should be translated as "region", while if it is used to denote a scientific concept, the translation should be "domain". SALAT uses the logical deduction rules shown below to make this choice:

1. <U (UQ (◇C(◇F N(□D C1 (◇E ($GEBIET))) ◇Y)SIT (△B)) ZEIT (△A))>.
   DAB.:<U (UQ (S (N (□D C1 ($GEBIET)) C1 ($GEOGR)) SIT (△B)) ZEIT (△A))>.

   →

<U (UQ (◇C (◇F N (□D C1 (◇E ($RE1GION))) ◇Y) SIT (△B)) ZEIT (△A))>.

2. <U(UQ(◇C (◇F N(□D C1 (◇E ($GEBIET)))◇Y) SIT (△B))ZEIT(△A))>.
   DAB.:<U (UQ (S (N (□D C1 ($GEBIET)) C1 ($WISSENSCH)) SIT (△B)) ZEIT (△A))>.

   →

   <U (UQ (◇C (◇F N (□D C1 (◇E (DOMAINE))) ◇Y) SIT (△B)) ZEIT (△A))>.

The difficulty with SALAT as a basis for a practical system seems, however, to be in the heavy use of logical deduction. If SALAT tries to translate a nontrivial fragment of a language, it will have to store and utilize a very large number of logical deduction rules. Accordingly, it must incorporate a sophisticated mechanism to access to a large deductive data base. Use of logical deduction to make inferences in language translation itself is very interesting and seems to be useful. But the point we want to stress is that they have to distinguish the kind of knowledge used in language translation. Most of the knowledge used in language translation seems to be utilized in a more straightforward manner. Such knowledge can be represented and stored using our functional application formalism in a more natural manner. Functional application formalism is not a problem solving formalism but it is simply a procedural model which can serve as a device for storing problem solving knowledge in a systematic way.

Advantage of our System. Thus neither Rosetta nor SALAT seems to utilize the most significant feature of Montague grammar: functionalism or compositionality. The advantage of our system is that it utilizes the functionality of Montague grammar intensively.

Answering Initial Questions

So far, we have answered one of our initial questions:

> How can we build a machien translation system based on Montague grammar?

Now we attempt to answer the remaining question:

> Is Montague garmmar useful in building a machine translation system?

We answer this question by evaluating our prototype system in terms of our initial design goals.

Richness of Expressive Power of Semantic Representation. In EFR, we can make a very wide range of distinctions in a natural manner as described in chapter 3. The limitation is, however, inability to represent discourse issues. Discourse

structure such as topic structure or focus is considered to be key issues in text understanding[Schank 77]. To take this aspect into consideration, it would be better to generate target language sentences from semantic networks. We ourselves did not take this approach simply because long-range research is thought to be needed in building an algorithm for generating discourse structure from memory structure.

Conciseness. Conciseness of the system is very good. The core of the machine translation system is a number of very small rule interpreters for each formal language we used. Actually, we have faced few coceptual difficulties in writing rules for translation. The problem is that we allow tree tranformation rules in CPSF. The use of transformation rules is against our initial intention, since we wanted to avoid computational complexity arising from allowing unrestricted use of tree-tree transformations. We attempted to replace tree-tree transformation by one-directional application of lambda-conversion-and-nesting rules by introducing an ordering carefully designed to reflect the regularity of the language. This attempt has not succeeded completely, but it is worth-while to do research in this direction.

Simple Control Structure. As is discussed above, this goal was only partially attained. In addition to the existence of restricted tree-tree transformation rules, what we call the higher-order problem arises in the course of building the prototype. This is mainly due to the assymmetric features of data and function. That is, lambda notation is heavily oriented to function (or procedure). When we want to regard a lambda form as data, we have lots of difficulties. An object-oriented view is useful in dealing with this difficulty, since it stresses the symmetry of data and procedure.

Maximum Utility of Knowledge. As we have shown in chapter 5, a rule for choosing a target language word can suffice for dealing with various variations, for instance relative clauses, passives and the like. This implies both economy and integrity of the rules. But as to the sharing of the knowledge, we can only use the notion of subroutine calls to utilize knowledge encoded in other rules. A more flexible way of knowledge sharing is to use the notion of inheritance as used in dealing with semantic networks. In the future, this kind of resource sharing must also be incorporated in our system.

Exception Handling. Our exception handling facility is very powerful for several reasons. First, our system is lexically driven, allowing for a word specific rule to be written. Second, we can write any procedure for handling exceptions, since we can

assign any form of CPSF to each lexical item. The only requirement is type agreement, which is needed to preserve integrity. The general procedure is encoded as application rules of the category. The shortcoming of the application rule schema, however, is inability to share the partially common rules. This seems to be solvable using the same sort of technique as for semantic networks.

Directions for Future Research

Research on natural languages involve a great number of unsolved questions. We only point out three questions closely related to this research.

to improve the formal representation in such a way that it does not contain lambda forms. Lambda forms are a source of a number of technical difficulties in the current formalism.

to evaluate the semantic network model in a restricted domain. Ongoing research desribed in Nishida (83a and b) reflects this orientation.

to improve a man-machine interface of machine translation system. This aspect is stressed by Melby (80, 82, 83) and translaters engaged in machine translation projects[Lawson 82].

8. Conclusion

Montague grammar have provided us a fairly sophisticated formal tool for analyzing phenomena in natural languages. What we have done in this thesis is to build a computational model for machine translation based on the general idea introduced by Montague grammar.

We have applied the notion of functionality to language translation. We have built an experimental system by which the aspects of the proposal can be demonstrated. We have also investigated the semantic network model of natural language as a computational basis for the use of formal languages. We treated the semantics interpretation as a process of memory modification and we have described how pieces of semantic network are created and put together as a sentence is interpreted.

All we have proposed is following what we call the computational constraint: Any computation defined in any computational model for natural language should be computationally feasible; it should terminate in a reasonable time and the necessary memory space is reasonably small. By this constraint we mean that a reasonable model for natural language semantics should also have a component for performance as well as for competance.

What is not attained is to encode various linguistic knowledge into the new framework. This leaves a big and encouraging question in future.

References

[Allen 82]
Allen,J.F. and Frisch,A.M., What's in a Semantic Networks?, in *Proc. 20th Annual Meeting of the Association for Computational Linguistics*, 19-27, 1982.

[Bobrow 77]
Bobrow,D.G., Kaplan,R.M., Kay,M., Norman,D.A., Thompson,H., and Winograd,T., GUS: A Frame-Driven Dialog System, *AI* 8(1977), 155-173, 1977.

[Boitet 80]
Boitet,C. and Nedobejkine,N., Russian-French at GETA: Outline of the Method and Detailed Example, in *Proc. COLING 80*, 437-446, 1980.

[Charniak 81]
Charniak,E., Six Topics in Search of a Parser: An Overview of AI Language Research, in *Proc.IJCAI-81*, 1079-1087, 1981.

[Cresswell 73]
Cresswell,M.J., *Logics and Languages*, Methuen & Co.LTd., 1973, (translated into Japanese by ISHIMOTO and IKEYA, KINOKUNIYA, 1978).

[Doshita 78a]
Doshita,S., Hiramatsu,K. and Kakui,M., Implementation of LISP System using Direct Accessible Bulk Memory, in *Trans. IECE Japan*, (D), Vol. J61-D, No. 5, 360-361, 1978.

[Dowty 78]
Dowty,D., *A Guide to Montague's PTQ*, Indiana University Linguistic Club, 1978.

[Dowty 81]
Dowty,D., Wall,R. and Peters,JR, *Introduction to Montague Semantics*, Reidel, 1981.

[Fikes 77]
Fikes,R. and Hendrix,G., A Network-Based Knowledge Representation and its Natural Deduction System, in *Proc. IJCAI-77*, 235-246, 1977.

[Friedman 78a]
Friedman,J. and Warren,D.S., A Parsing System for Montague Grammar, *Linguistics and Philosophy* 2, 347-372, 1978.

[Friedman 78b]
Friedman,J. Moran,D. and Warren,D.S., Evaluating English Sentences in a Logical Model: A Process Version of Montague Grammar, Abstract 16, Information Abstracts, *COLING 78*, 1978.

[Friedman 79]
Friedman,J., An Unlabeled Bracketing Solution to the Problem of Conjoined Phrase in Montague's PTQ, *Journal of Philosophical Logic* 8, 151-169, 1979.

[Friedman 80]
Friedman,J. and Warren,D.S., Lambda-Normal Forms in an Intensional Logic for English, *Studia Logica* XXXIX, 311-324, 1980.

[Friedman 81]
Friedman,J., Expressing Logical Formulas in Natural Language, Abstracts and Program, in Groenendijk,J.A.G., Janssen,T.M.V. and Stokhof,M.B.J. (eds.), *Formal Methods in the Study of Language*, MC TRACT 135, 113-130.

[Gawron 82]
Gawron,J.M., *et al*, Processing English with a Generalized Phrase Structure Grammar, in *Proc. 20th Annual Meeting of the Association for Computational Linguistics*, 74-81, 1982.

[Gazder 83]
Gazder,G., Phrase Structure Grammars and Natural Languages, in *Proc. IJCAI-83*, 556-565, 1983.

[Habel 80]
Habel,C.U., Rollinger,C.R., Schmidt,A. and Schneider,H.J., A Logic Oriented Approach to Automatic Text Understanding, in Bolc,L. (ed.), *Natural Language Computer Systems*, Carl Hanser Verlag, 1980.

[Hauenschild 79]
Hauenschild,C., Huckert,E. and Maier,R., SALAT: Machine Translation Via Semantic Representation, in Bauerle et al. (eds.), *Semantics from Different Points of View*, Springer-Verlag, 324-352, 1979.

[Hobbs 78]
Hobbs,J.R. and Rosenschein,R.J., Making Computational Sense of Montague's Intensional Logic, *AI* 9(1978), 287-306, 1978.

[Hornby 74]
Hornby,A.S., *Oxford Advanced Learner's Dictionary of Current English*, Oxford University Press, 1974, (reprinted from KAITAKUSHA, 1980).

[Hornby 75]
Hornby,A.S., *Guide to Patterns and Usage in English*, Oxford University

Press, 1975, (translated into Japanese by Itoh,K., Tokyo Oxford University Press, 1977).

[Hutchins 78]
Hutchins,W., Progress in Documentation, Machine Translation and Machine Aided Translation, *Journal of Documentation*, Vol. 34, No. 2, 119-159, 1978.

[Indurkhya 81]
Indurkhya,B., Sentence Analysis Programs based on Montague Grammar, Thesis submitted to the Netherlands Universities Foundation for International Cooperation.

[Ishihara 74a]
Ishihara,Y. and Tamati,T., On the D-tree Model and Language Analysis for English-Japanese Machine Translation Based on the Model, *Trans. IECEJ*, Vol. 57-D, No. 7, 435-442, 1974.

[Ishihara 74b]
Ishihara,Y. and Tamati,T., On an English-Japanese Machine Translation System Based on the D-tree Model and Its Experiment, *Trans. IECEJ*, Vol. 57-D, No. 7, 443-450, 1974.

[Janssen 77]
Janssen,T,M.V. and Boas,V.E., On the Proper Treatment of Referencing, Dereferencing and Assignment, in Salomaa,A. and Steinby,M. (eds.), *Automata, Languages, and Programming (proc. 4th Coll. Turku)*, Lecture Notes in Computer Science 52, Springer Verlag, Berlin, 282-300, 1977.

[Janssen 78]
Janssen,T.M.V. and Boas,V.E., The Expressive Power of Intensional Logic in the Semantics of Programming Languages, in Grusaka,J. (ed.), *Mathematical Foundation of Computer Science*, 1977 (*Proc. 6th Symp. Tatranska Lomnica*), Lecture Notes in Computer Science 53, Springer Verlag, Berlin, 303-311, 1977.

[Janssen 80a]
Janssen,T.M.V., Logical Investigations on PTQ arising from Programming Requirements, *Synthese* 44, 361-390, 1980.

[Janssen 80b]
On Intensionality in Programming Languages, in Heny,F.(ed.), *Ambiguities in Intensional Contexts*, Synthese Library, Reidel, 253-269, 1980.

[Janssen 80c]
Janssen,T.M.V., On Problems Concerning the Quantification Rules in Montague Grammar, in Rohrer,C.(ed.), *Time, Tense, and Quantification, Proc. of the Stuttgart Conference on the Logic of Tense and Quantification*, Max Niemeyer Verlag, 113-134, 1980.

[Janssen 81]
Janssen,T.M.V., Compositional Semantics and Relative Clause Formation in Montague Grammar, in Groenendijk,J.A.G., Janssen,T.M.V. and Stokhof,M.B.J. (eds.), *Formal Methods in the Study of Language*, MC TRACT 135, 237-276, 1981.

[Kaplan 82]
Kaplan,R.M. and Bresnan,J., Lexical-Functional Grammar: A Formal System for Grammatical Rerpesentation, in Bresnan (ed.), *The Mental Representation of Grammatical Relations*, 173-281, The MIT Press, 1982.

[Landsbergen 80]
Landsbergen,J., Adaptation of Montague Grammar to the Requirement of Question Answering, in *Proc. COLING 80*, 211-212, 1980.

[Landsbergen 81]
Adaptation of Montague Grammar to the Requirements of Parsing, in Groenendijk,J.A.G., Janssen,T.M.V. and Stokhof,M.B.J. (eds.), *Formal Methods in the Study of Language*, MC TRACT 136, 1981.

[Landsbergen 82]
Machine Translation based on Logically Isomorphic Montague Grammars, in *Proc. COLING 82*, 1982.

[Lawson 82]
Lawson,V. (ed.), *Practical Experience of Machine Translation, Proceedings of a Conference*, London, 5-6 November 1981, North-Holland, 1982.

[Lehmann 81]
Lehmann,W.P., *et al*, *The Metal System*, Rome Air Development Center, RADC-TR-90-374, Vol I, Vol II, 1981.

[Marcus 80]
Marcus,P.M., *A Theory of Syntactic Recognition for Natural Language*, The MIT Press, 1980.

[Matsumoto 81]
Software Implementation of Montague Grammar and Related Problems in Iguchi,S. (ed.), Formal Approaches to Natural Language, Proc. of the First Colloquium on Montague Grammar and Related Topics, 148-158, 1981.

[Melby 80]
Melby,A.K., Smith,M.R. and Peterson,J., ITS: Interactive Translation System, in *Proc.COLING 80*, 424-429, 1980.

[Melby 82]
Melby,A.K., Multi-Level Translation Aids in a Distributed System, in Horecky (ed.), *COLING 82: Proceedings of the Ninth International Conference on Computational Linguistics*, 215-220, 1982.

[Melby 83]
Melby,A.K., Computer Assisted Translation Systems: The Standard Design

and a Multi-Level Design, in *Proc. Applied Natural Language Processing*, 174-177, 1983.

[Montague 74a]
Montague,R., Universal Grammar, in Thompson (ed.), *Formal Philosophy*, Yale University, 222-246, 1974.

[Montague 74b]
Montague,R., Proper Treatment of Quantification in Ordinary English, in Thompson (ed.), *Formal Philosophy*, Yale University, 247-270, 1974.

[Moore 77]
Moore,R.C., Reasoning about Knowledge and Action, in *Proc. IJCAI-77*, 223-227, 1977.

[Nagao 77]
Nagao,M. and Tsujii,J., Programs for Natural Language Processing, *J. of IPSJ*, Vol. 18, No.1, 63-75, 1977.

[Nagao 78]
Nagao,M., Understanding of Natural Language, *J. of IPSJ*, Vol. 19, No. 10, 952-961, 1978.

[Nagao 79a]
Nagao,M., Information Science and Natural Language, *J. of IPSJ*, Vol. 20, No. 3, 176-183, 1979.

[Nagao 79b]
Nagao,M., Machine Translation, *J. of IPSJ*, Vol. 20, No. 10, 896-902, 1979.

[Nagao 80]
Nagao,M., Tsujii,J., Mitamura,K., Hirakawa,H. and Kume,M., A Machine Translation System from Japanese into English -- Another Perspective of MT system --, in *Proc.COLING 80*, 414-423, 1980.

[Nagao 82]
Nagao,M., Tsujii,J., Yada,K. and Kakimoto,T., An English Japanese Machine Translation System of the Titles of Scientific and Engineering Papers, in Horecky (ed.), *COLING 82: Proceedings of the Ninth International Conference on Computational Linguistics*, 245-252, 1982.

[Nilsson 80]
Nilsson,N.J., *Principles of Artificial Intelligence*, Tioga, 1980.

[Nishida(F.) 80]
Nishida,F., Takamatsu,S. and Kuroki,H., English-Japanese Translation Through Case-Structure Conversion, in *Proc. COLING 80*, 447-454, 1980.

[Nishida(F.) 82]
Nishida,F. and Takamatsu,S., Japanese-English Machine Translation through Internal Expressions, in Horecky (ed.), *COLING 82: Proceedings of the Ninth International Conference on Computational Linguistics*, 271-276, 1982.

[Nishida 83a]
Nishida,T., Kosaka,A., and Doshita,S., Information Extraction from Technical Descriptions -- Analysis of Hardware Manuals, in *Proc. Symposium on Natural Language Processing Technology*, June 16-17, 1983, 101-110, IPSJ, (in Japanese).

[Nishida 83b]
Nishida,T., Kosaka,A., and Doshita,S., Towards Knowledge Acquisition from Natural Language Documents -- Automatic Model Construction from Hardware Manual --, in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI 83)*, 482-486, 1983.

[Nitta 82]
Nitta,Y., Okajima,A., Yamano,F. and Ishihara,K., A Heuristic Approach to English-into-Japanese Machine Translation, in Horecky (ed.), *COLING 82: Proceedings of the Ninth International Conference on Computational Linguistics*, 277-282, 1982.

[Pereira 80]
Pereira,F. and Warren,D., Definite Clause Grammars for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Networks, *AI* 13(1980), 231-278.

[Quillian 68]
Quillian,R., Semantic Memory, in Minsky (ed.), *Semantic Information Processing*, The MIT Press, 1968.

[Quirk 73]
Quirk and Green, *University Grammar*, translated into Japanese from KINOKUNIYA-SHOTEN, 1977.

[Rieger 79]
Rieger,C. and Small,S., Word Expert Parsing, in *Proc. IJCAI-79*, 723-728, 1979.

[Robinson 80]
Robinson,J.J., DIAGRAM: A Grammar for Dialogues, SRI Technical Note 205, 1980.

[Rosenschein 82]
Rosenschein,S.J. and Schieber,S.M., Translating English into Logical Form, in *Proc. 20th Annual Meeting of Association for Computational Linguistics*, 1-8, 1982.

[Sager 81]
Sager,N., *Natural Language Information Processing, A Computer Grammar of English and its Applications*, Addison-Wesley, 1981.

[Sakai 66]
Sakai,T. and Sugita,S., Mechanical Translation of English inot Japanese, *J. of IECEJ*, Vol. 49, No. 2, 46-53, 1966.

[Sakai 69]
Sakai,T., Sugita,S. and Watanabe,A., Mechanical Translation from Japanese into English, *J. of IPSJ*, Vol. 10, No. 6, 418-427, 1969.

[Scha 80]
Scha,R.J.H., Schoenmakers,W.J. and Van Utteren,E.P.C., The Question Answering System PHLICA1, In Bolc (ed.), *Natural Language Question Answering Systems*, MaCmillan, 1982.

[Schank 77]
Schank,R.C. and Abelson,R.P., *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*, Lawrence Erlbaum Associates, 1977.

[Schubert 76]
Schubert,L.K., Extending the Expressive Power of Semantic Networks, *AI*, Vol. 7, No. 2, 1976.

[Shudo 77]
Shudo,K., Tsurumaru,H. and Yoshida,S., A Predicative Part Processing System for Japanese-English Machine Translation, *Trans. IECEJ*, Vol. J60-D, No. 10, 830-837, 1977.

[Simmons 73]
Simmons,R.F., Semantic Network: Their Computation and Use for Understanding English Sentences, in Schank, *et al* (eds), *The Computer Models of Thought and Language*, Freeman and Company, 63-113, 1973.

[Slocum 83]
Slocum,J., A Status Report on the LRC Machine Translation System, in *Proc. Applied Natural Language Processing*, 166-173, 1983.

[Sugita 68]
Sugita,S., *A Study on Mechanical Translation from English into Japanese*, Doctoral Thesis, Kyoto University, 1968.

[Tanaka 77]
Tanaka,H., Sato,T. and Motoyoshi,F., A Programming System for Natural Language Processing -- On Extended LINGOL --, in *Trans. IECEJ*, Vol. J60-D, No. 12, 1061-1068, 1977.

[Tsujii 78]
Tsujii,J., *A Computational Model of Natural Language Understanding for Japanese*, Doctoral Thesis, Kyoto University, 1978.

[Uchida 80]
Uchida,H. and Sugiyama,K., A Machine Translation System from Japanese into English Based on Conceptual Structure, in *Proc. COLING 80*, 455-462, 1980.

[Walker 78]
Walker,D. (ed.), *Understanding Spoken Language*, North-Holland, 1978.

[Winograd 72]
Winograd,T., *Understanding Natural Language*, Academic Press, 1972.

[Winograd 83]
Winograd,T., *Language as a Cognitive Process, Volume 1: Syntax*, Addison-Wesley, 1983.

[Woods 70]
Woods, W.A., Transition Network Grammar for Natural Language Analysis, *Comm. ACM*, Vol. 13, 1970.

[Woods 75]
Woods,W.A., What's in a Link: Foundations for Semantic Networks, in Bobrow and Collins (eds), *Representation and Understanding, Studies in Cognitive Science*, Academic Press, 1975.

[Yoshida 79]
Yoshida,S., Hierarchical Concepts Structure for Natural Language Understanding System, in *Proc.IJCAI-79*, 1001-1003, 1979.

[Yonezaki 80a]
Yonezaki,N. and Enomoto,H., Data Base System based on Intensional Logic, in *Proc. COLING 80*, 220-227, 1980.

[Yonezaki 80b]
Yonezaki,H. and Enomoto,H., Formalization of Database with Hierarchical Data based on Intensional Logic, Technical Report, Dept. of Computer Science, Tokyo Institute of Technology, CS-Y8002, 1980.

Publications

Main Publications

1. Nishida,T. and Doshita,S., The Framework of Knowledge Representation and its Retrieval in LGS -- The Literature Guide System, in *Proceedings of the Sixth International Joint Conference on Artificial Intelligence (IJCAI 79)*, 662-664, 1979.

2. Nishida,T. and Doshita,S., Hierarchical Meaning Representation and Analysis of Natural Language Documents, in *Proceedings of Eighth International Conference on Computational Linguistics (COLING 80)*, 85-92, 1980.

3. Nishida,T. and Doshita,S., A Knowledge-based Literature Guide System -- A New Approach to Document Retrieval, in Lavington(ed.), *Information Processing 80 (Proceedings of IFIP Congress 80)*, 699-704, North-Holland Publishing Company, 1980.

4. Nishida,T., Kiyono,M. and Doshita,S., An English-Japanese Machine Translation System Based on Montague Grammar, in *Trans. IPSJ*, Vol. 23, No. 2, 1982, (in Japanese).

5. Nishida,T. and Doshita,S., COMPLAN: A Programming System for Natural Language Analysis, in *Trans. IPSJ*, Vol. 23, No. 4, 1982, (in Japanese).

6. Nishida,T. and Doshita,S., An English-Japanese Machine Translation System based on Formal Semantics of Natural Language, in Horecky, J. (ed.), *COLING 82 -- Proceedings of Ninth International Conference on Computational Linguistics*, 277-282, North-Holland Publishing Company, 1982.

7. Nishida,T. and Doshita,S., An Application of Montague Grammar to English-Japanese Machine Translation, in *Proceedings of Conference on Applied Natural Language Processing*, 1-3 February 1983, Miramar-Sheraton Hotel, 156-165, 1983.

8. Nishida,T. and Doshita,S., A New Approach to Machine Translation Using Montague Grammar, *Bit* Vol.15, No.3, 231-247, 1983, (in Japanese).

9. Nishida,T., Kosaka,A., and Doshita,S., Information Extraction from Technical Descriptions -- Analysis of Hardware Manuals, in *Proc. Symposium on Natural Language Processing Technology*, June 16-17, 1983, 101-110, IPSJ, (in Japanese).

10. Nishida,T., Kosaka,A., and Doshita,S., Towards Knowledge Acquisition from Natural Language Documents -- Automatic Model Construction from Hardware Manual --, in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI 83)*, 482-486, 1983.

11. Doshita,S. and Nishida,T., An English-Japanese Machine Translation System Through Functional Semantic Processing, in Tanaka,K. (ed.), *Knowledge Engineering*, Asakura-Shoten, (to appear), (in Japanese).

Other Publications

12. Nishida,T. and Yamasaki,S., Basic Consideration on a Mechanism for Deduction and Retrieval Using Metric Relationship among Knowledge Representations, in *Proc. of Joint Convention of IECEJ*, No. 1106, 1977, (in Japanese).

13. Yamasaki,S. and Nishida,T., On Distance between Tree Structures, in *Proc. of Joint Convention of IECEJ*, No. 1107, 1977, (in Japanese).

14. Doshita,S. and Nishida,T., Bibliographic Information Retrieval Based on Abstracts, Technical Report of the Professional Group on Automaton and Language of IECEJ, AL-77-76, 1978, (in Japanese).

15. Doshita,S. and Nishida,T., Knowledge Representation for Abstracts, in *Proc. 19th Convention Record of IPSJ*, 4E-6, 1978, (in Japanese).

16. Doshita,S. and Nishida,T., A Program for Analyzing English Sentences using Procedural Rules, in *Proc. Conventioanl Record of IECEJ*, No. 1207, 1979, (in Japanese).

17. Nishida,T. and Doshita,S., Knowledge Representaiton and Organization of the Knowledge Base for a Literature Guide System LGS, in *Proc. 20th Convention Record of IPSJ*, 4C-7, 1979, (in Japanese).

18. Nishida,T., Kiyono,M. and Doshita,S., A Proposal for Processing Interrogative Sentences of Japanese, in *Proc. 20th Convention Record of IPSJ*, 5C-2, 1979, (in Japanese).

19. Nishida,T. and Doshita,S., A Guide System for Scientific Literature based on Hierarchical Representaiton of Natural Language Semantics, Technical Report of the Professional Group on Automaton and Language of IECEJ, AL-79-95, 1980, (in Japanese).

20. Nishida,T. and Doshita,S., Construction of a Parser for Analyzing English Sentences (EASY) -- Algorithm for Syntactic Analysis --, in *Proc. Conventioanl Record of IECEJ*, No. 1198, 1980, (in Japanese).

21. Nishida,T., Kiyono,M., Yamanaka,T. and Doshita,S., Experiments for Extracting Semantic Representation using a Parser for Analyzing English Sentences (EASY), in *Proc. Conventioanl Record of IECEJ*, No. 1199, 1980, (in Japanese).

22. Nishida,T. and Doshita,S., LOGICS: Hierarchical Meaning Representation System for Natural Language and its Application, Technical Report of WG on Computational Linguistics of IPSJ, 22-2, 1980, (in Japanese).

23. Nishida,T., Kiyono,M., Yamanaka,T. and Doshita,S., English-Japanese Machine Translation based on Semantic Analysis, in *Proc. 21th Convention Record of IPSJ*, 5H-1, 1980, (in Japanese).

24. Nishida,T., Sakakibara,Y. and Doshita,S., Analysis of Japanese Predicatives and Their Translation into English, in *Proc. 21th Convention Record of IPSJ*, 5H-3, 1980, (in Japanese).

25. Nishida,T. and Doshita,S., On a Knowledge Data Base based on the Formal Semantics for Natural Language, in *Proc. 22th Convention Record of IPSJ*, 5M-7, 1981, (in Japanese).

26. Nishida,T, Kiyono,M. and Doshita,S., An English-Japanese Machine Translation System Using Semantic Analysis, in *Proc. 22th Convention Record of IPSJ*, 2M-2, 1981, (in Japanese).

27. Nishida,T., Kosaka, A. and Doshita,S., A Japanese-English Machine Translation System with a Logical Language as Intermediate Language, in *Proc. 22th Convention Record of IPSJ*, 2M-3, 1981, (in Japanese).

28. Nishida,T. and Doshita,S., Generating Japanese Sentences from Logical Forms of Intensional Logic, in *Proc. 23th Convention Record of IPSJ*, 4M-12, 1981, (in Japanese).

29. Nishida,T. and Doshita, S., An English-Japanese Machine Translation System based on Montague Grammar, Technical Report of WG on Artificial Intelligence and Interactive Techniques of IPSJ, 23-2, 1981, (in Japanese).

30. Nishida,T., Kosaka,A. and Doshita,S., On the Improvement of an English-Japanese Machine Translation System based on Montague Grammar, in *Proc. 24th Convention Record of IPSJ*, 4K-6, 1982, (in Japanese).

31. Nishida,T., Kosaka,A. and Doshita,S., Report on the Current Status of an English-Japanese Machine Translation System based on Montague Grammar, in *Proc. 24th Convention Record of IPSJ*, 4K-5, 1982, (in Japanese).

32. Yang,Y., Nishida,T. and Doshita,S., Consideration on a Chinese Language Analysis System, Technical Report of WG on Computational Linguistics of IPSJ, 33-1, 1982, (in Japanese).

33. Nishida,T, Kosaka,A. and Doshita,S., Consideration on a Scientific Document Understanding System, in *Proc. 25th Convention Record of IPSJ*, 7H-4, 1982, (in Japanese).

34. Nishida,T., Kosaka,A. and Doshita,S., On the Generation Phase of an English-Japanese Machine Translation System based on Montague Grammar, in *Proc. 25th Convention Record of IPSJ*, 6K-5, 1982, (in Japanese).

35. Yang,Y., Nishida,T. and Doshita,S., Consideration on the Problems on Chinese Language Analysis, in *Proc. 25th Convention Record of IPSJ*, 6K-7, 1982, (in Japanese).

36. Nishida,T., Kosaka,A. and Doshita,S., Consideration on the Automatic Retrieval of Information from Hardware Manuals, Technical Report of the Professional Group on Automaton and Language of IECEJ, AL-82-68, 1982, (in Japanese).

37. Nishida,T., Kosaka,A. and Doshita,S., Natural Language Analysis for Understanding Hardware Manuals, in *Proc. 26th Convention Record of IPSJ*, 6L-1, 1983, (in Japanese).

38. Nishida,T., Kosaka,A. and Doshita,S., Description Model and Inference Method for Understanding Hardware Manuals, in *Proc. 26th Convention Record of IPSJ*, 7C-1, 1983, (in Japanese).

39. Nishida,T., Basic Design of the Transfer Stage, Technical Report of WG on Computational Linguistics of IPSJ, 38-6, 1983, (in Japanese).

IECEJ : The Institute of Electronics and Communication Engineers of Japan

IPSJ : Information Processing Society of Japan.

Appendix: Translation of Sample Sentences

This appendix shows the result of the translation of four sample texts (including forty sentences). All of these texts were taken from scientific and engineering field. For the reader's convenience, the summary of the result is shown first, then the details are shown. Note that these translations were actually carried out in a sentence-by-sentence manner and then hand-assembled, although the summary might give the reader an impression as if the translation was carried out text-by-text.

Each of detailed descriptions includes: the result of phrase structure analysis, an extracted EFR expression, and a phrase structure synthesized from the EFR expression, as well as input and output strings. Each of them is given a uniquely identifiable serial number from S-1 to S-40. The reader can find these sentence numbers at the rightmost column of the summary, by means of which the reader can refer to the detailed description of each sample translation.

Note that the internal representation of each sample sentence is slightly different from the original sentence; all letters are replaced by upper case letters, and commas and periods are replaced by symbols ' * ' and ' / ', respectively.

Citation of the source texts:

Text-1: Zilog, Z80-Assembly Language Programming Manual, Introduction, 1977.

Text-2: Metcalfe,R.M. and Boggs,D.R., Ethernet: Distributed Packet Switching for Local Computer Networks, Abstract, CSL-75-7, Xerox Palo Alto Research Center, 1975.

Text-3: Metcalfe,R.M. and Boggs,D.R., Ethernet: Distributed Packet Switching for Local Computer Networks, 2. System Sammary, CSL-75-7, Xerox Palo Alto Research Center, 1975.

Text-4: Motorola: VERSAbus Preliminary Specification, 2.1 Multiple Processor Philosophy, 1979.

(Each of these texts is reproduced.)

Summary of the Results

Translation of Text-1:

Input:

The assembly language provides a means for writing a program without having to be concerned with actual memory addresses or machine instruction formats. It allows the use of symbolic codes (opcodes and operands) to represent the instructions themselves. Labels (symbols) can be assigned to a particular instruction step in a source program to identify that step as an entry point for use in subsequent instructions. Operands following each instruction represent storage locations, registers, or constant values. The assembly language also includes assembler directives that supplement the machine instruction. A pseudo-op, for example, is a statement which is not translated into a machine instruction, but rather is interpreted as a directive that controls the assembly process. 1 2 3 4 5 6

A program written in assembly language is called a source program. It consists of symbolic commands called statements. Each statement is written on a single line and may consists of from one to four entries: A label field, an operation field, an operand field and a comment field. The source program is processed by the assembler to obtain a machine language program (object program) that can be executed directly by the Z80. 7,8 9 10

Output:

そのアセンブリ言語は実際のメモリー番地または機械命令形式について関与させられなければならないことなしにプログラムを書くための手段を提供する。それはメモリーロケーションを同定するための記号番地とその命令自身を表現するためのニモニックコード(命令コードとオペランド)の使用を許す。ラベル(記号)はそのステップを後続の命令における使用のためのエントリーポイントとして同定するためにソースプログラムにおける特定の命令ステップに割り当てられることができる。それぞれの命令に従うオペランドはストレージロケーションまたはレジスタまたは定数の値を表現する。そのアセンブリ言語はまたその機械命令を補うアセンブラ命令を含む。擬似命令は例えば機械命令に翻訳されないが、むしろそのアセンブリ過程を制御する命令と解釈されるステートメントである。 1' 2' 3' 4' 5' 6'

そのアセンブリ言語において書かれるプログラム はソースプログラムと呼ばれる。それはステートメントと呼ばれる記号コマンドから成る。それぞれのステートメントは単一の行の上に書かれ、1から4までのエントリー(ラベルフィールドと操作フィールドとコメントフィールド)から成るかもしれない。そのソースプログラムはそのZ80によって直接実行されることができる機械言語プログラム(オブジェクトプログラム)を得るためにそのアセンブラによって処理される。 7' 8',9' 10'

Translation of Text-2

Input:

Ethernet is a branching broadcast communication system for carrying digital data packets among locally distributed computing stations. The packet transport mechanism provided by Ethernet has been used to build systems which can be either local computer networks or loosely coupled multiprocessors. 11 12

An Ethernet's shared communication facility, its Ether, is a passive broadcast medium with no central control. Coordination of access to the Ether for packet broadcasts is distributed among the contending transmitting stations using controlled statistical arbitration. Switching of packets to their destinations on the Ether is distributed among the receiving stations using packet address recognition. 13 14 15

Design principles and implementation are described based on experience with an operating ethernet of 100 nodes along a kilometer of coaxial cable. A model for estimating performance under heavy loads and a packet protocol for error-controlled communication are included for completeness. 16 17

Output:

ETHERNETはデジタルデータパケットを局所的に分散される計算ステーションの間に伝搬するための分岐状の放送通信システムである。局所的な計算機ネットワークまたは疎に結合されるマルチプロセッサとみなされることができるシステムを構築するためにETHERNETによって提供されるパケット輸送機構は用いられてしまう。 11' 12'

ETHERNETの共有された通信設備(そのETHER)は中央の制御を持たない受動性の放送媒体である。パケット放送のためのそのETHERへのアクセスの調整は制御される統計的調停を用いて競合している送信側のステーションに分散される。そのETHERの上の目的地へのパケットの交換はパケット番地認識を用いて受信側のステーションの間に分散される。 13' 14' 15'

設計原則と実働化は1Kmのコアキシャルケーブルに沿った100のノードの稼動中のETHERNETについての経験に基づいて述べられる。重い負荷の下での性能を評価するためのモデルと誤り制御された通信のためのパケットプロトコルは完全のために含まれる。 16' 17'

Translation of Text-3 (1/2)

Input:

Ethernet is a system for local communication among computing stations. Our experimental Ethernet uses tapped coaxial cables to carry variable-length digital data packets among, for example, personal minicomputers, printing facilities, large file storage devices, magnetic tape backup stations, larger central computers, and longer-haul communication equipment. 18,19

The shared communication facility, a branching Ether, is passive. A station's Ethernet interface connects bit-serially through an interface cable to a transceiver which in turn taps into the passing Ether. A packet is broadcast onto the Ether, is heard by all stations, and is copied from the Ether by destinations which select it according to the packet's leading address bit. This is broadcast packet switching and should be distinguished from store-and-forward packet switching in which routing is performed by intermediate processing elements. To handle the demands of growth, an Ethernet can be extended using packet repeaters for signal regeneration, packet filters for traffic localization, and packet gateway for internetwork address extension. 20,21 22 23 24

Output:

ETHERNETは計算ステーションの間の局所的な通信のためのシステムである。我々の実験的なETHERNETは可変長のデジタルデータのパケットを例えば個人のミニコンピュータと印刷設備と大きなファイルのストレージの装置と磁気テープバックアップステーションともっと大きな中央の計算機と長距離の通信設備の間に伝搬するためにタップを付けられたコアキシャルケーブルを用いる。 18',19'

共有された通信設備(分岐状のETHER)は受動性のものである。ステーションのETHERNETインタフェースは順に往来するETHERにタップインされる送受信機にインタフェースケーブルを介してビット直列に接続される。パケットはそのETHERの上に放送され、全てのステーションによって聞かれ、そのパケットの先導する番地bitに従ってそれを選択する目的地によってそのETHERから複写される。これは放送パケット交換であり、中間の処理要素によって経路決定が実行されるストア·アンド·フォワードパケット交換と区別されなければならない。増大の要求を取り扱うために信号再生のためのパケットリピータと交通局所化のためのパケットフィルターとネットワーク間のアドレス拡張のためのパケットゲートウェイを用いてETHERNETは拡張されることができる。 20',21' 22' 23' 24'

Translation of Text-3 (2/2)

Input:

Control is completely distributed among stations with packet transmissions coordinated through statistical arbitration. Transmissions initiated by a station defer to any which may already be in progress. Once started, if interference with other packets is detected, a transmission is aborted and rescheduled by its source station. After a certain period of interference-free transmission, a packet is heard by all stations and will run to completion without interference. Ethernet controllers in colliding stations each generate random retransmission intervals to avoid repeated collisions. The mean of a packet's retransmission intervals is adjusted as a function of collision history to keep Ether utilization near the optimum with changing network load. 25 26 27 28 29 30

Even when transmitted without source-detected interference, a packet may still not reach its destination without error; thus, packets are delivered *only with high probability.* Stations requiring a residual error rate lower than that provided by the bare Ethernet packet transport mechanism must follow mutually agreed upon packet protocols. 31 32

Output:

制御は統計的調停を介して調整されるパケット伝送で完全にステーションの間に分散される。ステーションによって起動される伝送は任意の既に進行におけるものであるかもしれないものに従う。一度伝送が開始されもし他のパケットについての妨害が検出されるならばそのソースステーションによって伝送は異常終了させられ、再スケジュールされる。妨害のない伝送の或る期間の後にパケットは全てのステーションによって聞かれ、妨害なしに完了に至るだろう。それぞれの衝突するステーションにおけるETHERNETコントローラは繰り返される衝突を避けるためにランダムな再送間隔を生成する。 25' 26' 27' 28' 29'

たとえソース側で検出された妨害なしにパケットが伝送されても依然パケットはエラーなしにその目的地に到着しないかもしれない;このように高い確率によってのみパケットが配達される。裸のETHERNETパケット輸送機構によって提供されるものよりも低い残余のエラー割合を要求するステーションは相互に同意されるパケット通信規約に従わなければならない。 30' 31'

Translation of Text-4

Input:

VERSAbus incorporates state-of-the-art features to support system architecture built around multiple microprocessors. Such architecture generates extremely high throughput. This may be achieved on the VERSAbus by bus arbitration and priority interrupt features that allow multiple masters on the VERSAbus. When future semiconductor devices are capable of supporting 100 nanosecond cycle times on data transfers, the 5 MHz transfer rate of the VERSAbus will be upgraded to permit data rates of 40 megabytes per second. 33 34 35 36

Furthermore, systems incorporating multi-processors may use several different varieties of processors. For example, a slow speed peripheral controller may be most economically designed using an 8-bit data controller to a system that may be supporting a 16- or 32-bit processor. VERSAbus provides a straightforward way of interfacing such an 8-bit data controller to a system that may be supporting a 16- or 32-bit processor. 37 38 39

The asynchronous operation of VERSAbus allows very slow peripherals to be matched to high speed processors without significant reduction to the overall system speed. 40

Output:

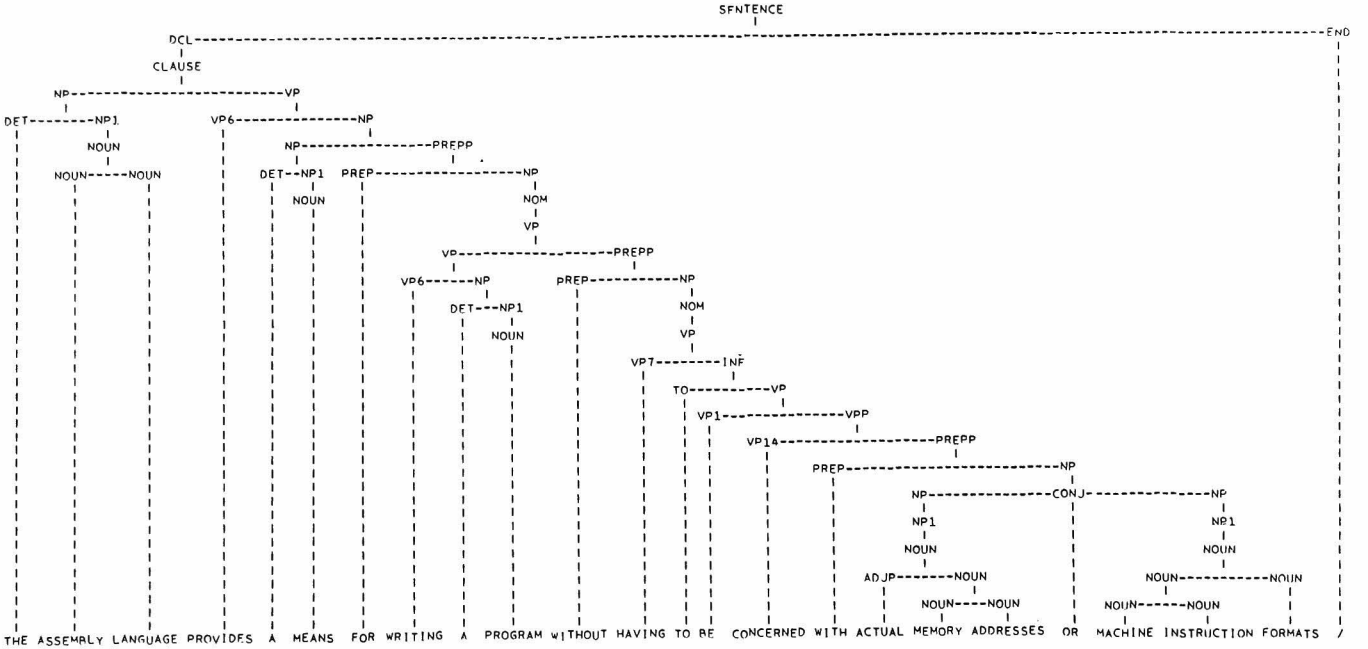
VERSAbusは多重マイクロプロセッサの周りに構築されるシステムアーキテクチャを支援するために最新の特性を組込む。そのようなアーキテクチャは極めて高いスループットを生成する。これはそのVERSAbusの上の多重マスターを許すバス調停及び優先度割込み特性によってそのVERSAbusの上に達成されるかもしれない。データ移動の上に将来の半導体装置が100ナノ秒サイクル時間を支援することが可能なものであるときそのVERSAbusの5MHz移動割合はそれぞれの秒につき40MBのデータ割合を許すために格上げされるだろう。 33' 34' 35' 36'

更にマルチプロセッサを組込むシステムはいくつかの異なる種類のプロセッサを用いるかもしれない。例えば8bitマイクロプロセッサを用いて最も経済的に遅い速度の周辺機器コントローラは設計されるかもしれない。VERSAbusはそのような8bitデータコントローラを16bitまたは32bitプロセッサを支援しつつあるかもしれないシステムとインターフェイスで接続することのまっすぐな方法を提供する。 37' 38' 39'

非常に遅い周辺機器はVERSAbusの非同期の操作によって全体のシステム速度への重大な縮小なしに高い速度のプロセッサに整合させられることが可能だ。 40'

Details of the Results

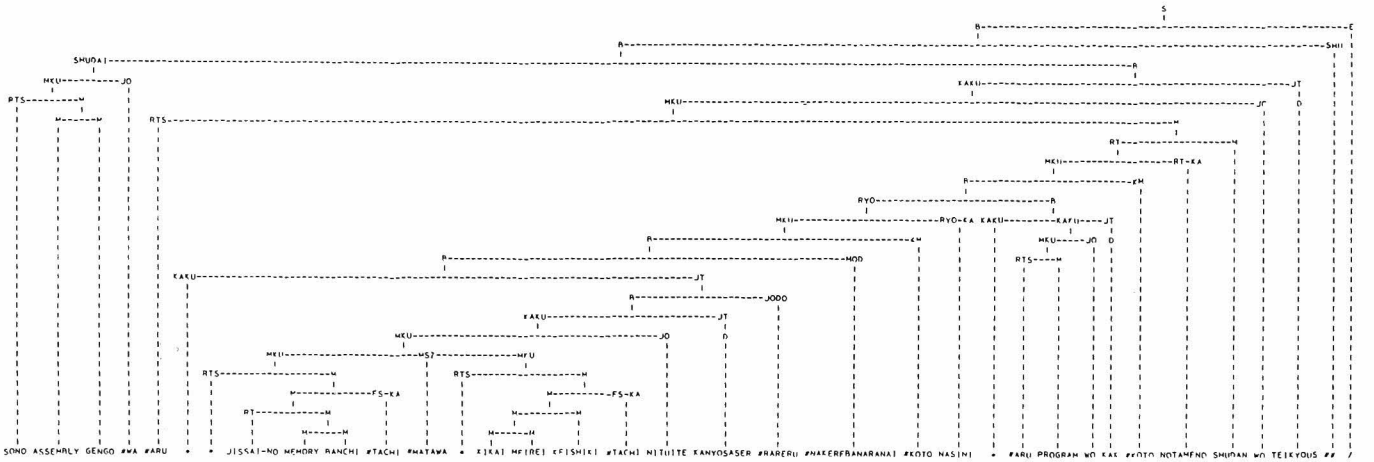
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((THE ((*ADJ-CLSF ASSEMBLY) LANGUAGE))
    (LAMBDA
      X186
      ((A
        (LAMBDA
          X185
          ((NOM
            (LAMBDA
              X179
              ((NOM
                (HAVE-TO
                  (LAMBDA
                    X163
                    ((*OR
                      (A* (*PL (ACTUAL ((*ADJ-CLSF MEMORY) ADDRESS))))
                      (A*
                        (*PL
                          ((*ADJ-CLSF ((*ADJ-CLSF MACHINE) INSTRUCTION))
                            FORMAT))))
                      (LAMBDA X164 ((*EN3 CONCERN-WITH) X163 X164))))))
                (LAMBDA
                  X178
                  ((WITHOUT X178)
                    ((A PROGRAM) (LAMBDA X152 (WRITE X179 X152))))))))
            (LAMBDA X184 ((((*AP FOR) X184) MEANS) X185)))))
        (LAMBDA X187 (PROVIDE X186 X187))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



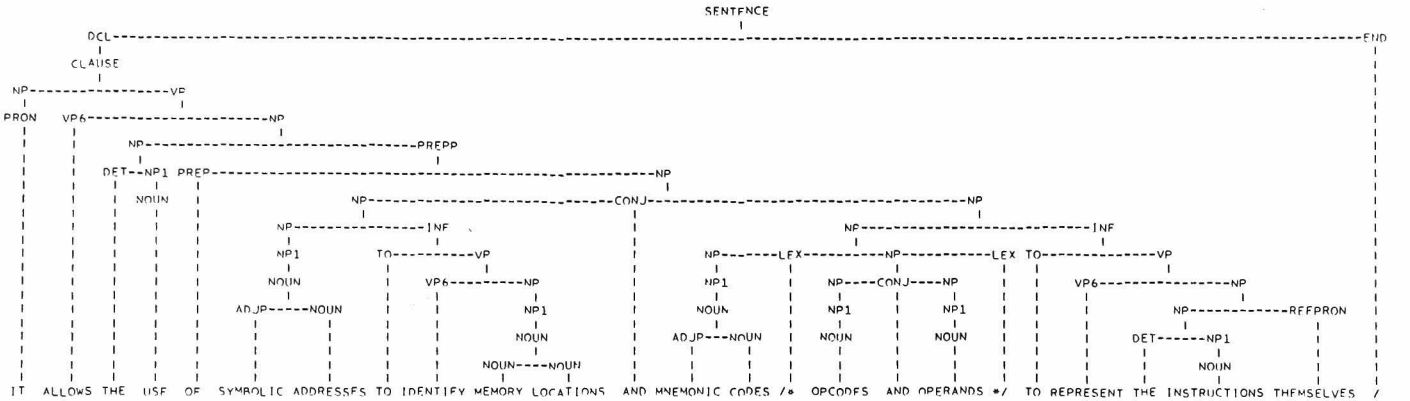
Input Sentence:

THE ASSEMBLY LANGUAGE PROVIDES A MEANS FOR WRITING A PROGRAM WITHOUT HAVING TO BE CONCERNED WITH ACTUAL MEMORY ADDRESSES OR MACHINE INSTRUCTION FORMATS /

Output Sentence:

そのアセンブリ言語は実際のメモリー番地または機械命令形式について関与させられなければならないことなしにプログラムを書くための手段を提供する。

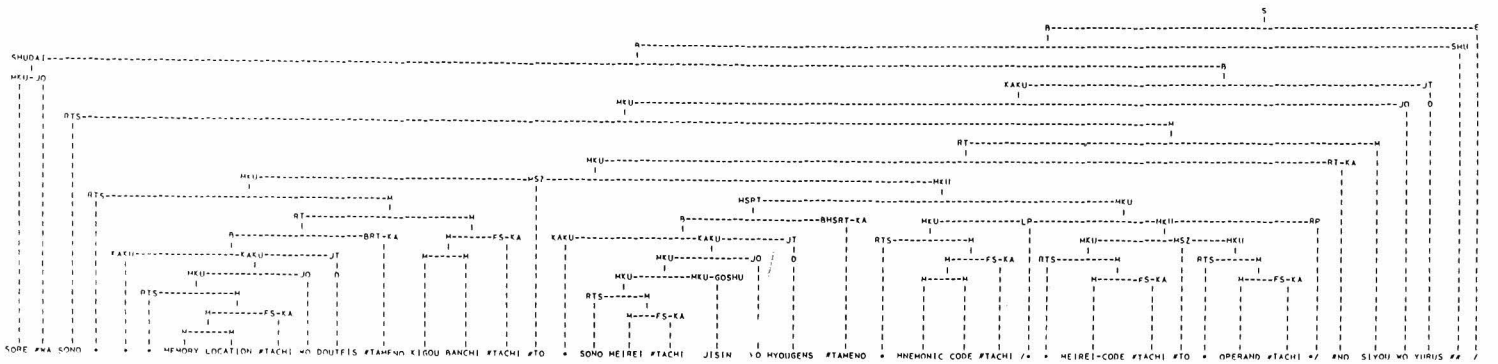
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (IT
    (LAMBDA
      X98
      ((THE
        (LAMBDA
          X87
          ((*AND
            (A*
              ((INF-ADJ
                (LAMBDA
                  X4
                  ((A* (*PL ((*ADJ-CLSF MEMORY) LOCATION)))
                    (LAMBDA X5 (IDENTIFY-A X4 X5)))))
                (*PL (SYMBOLIC ADDRESS))))
              ((INF-ADJ-NR
                (LAMBDA
                  X12
                  ((THEMSELVES (THE (*PL INSTRUCTION)))
                    (LAMBDA X13 (REPRESENT X12 X13)))))
                ((/**/ (*AND (A* (*PL OPCODE)) (A* (*PL OPERAND))))
                  (A* (*PL (MNEMONIC CODE))))))
            (LAMBDA X86 ((((*AP OF) X86) USE-N1) X87)))))
        (LAMBDA X99 (ALLOW X98 X99))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



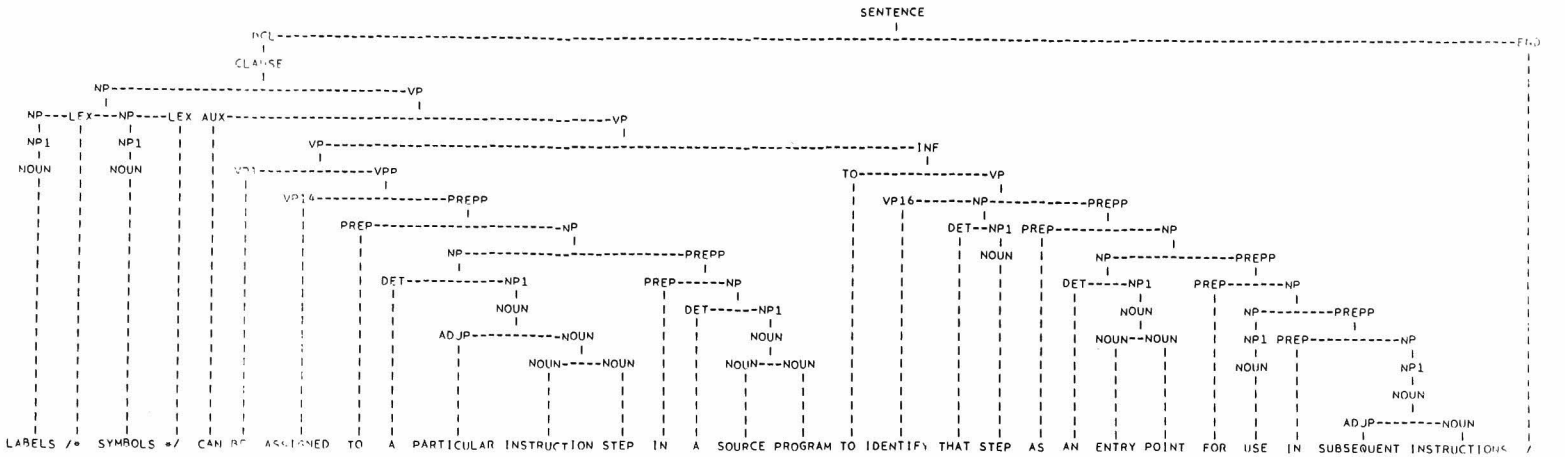
Input Sentence:

IT ALLOWS THE USE OF SYMBOLIC ADDRESSES TO IDENTIFY MEMORY LOCATIONS AND MNEMONIC CODES /* OPCODES AND OPERANDS */ TO REPRESENT THE INSTRUCTIONS THEMSELVES /

Output Sentence:

それはメモリーロケーションを同定するための記号番地とその命令自身を表現するための
ニモニックコード（命令コードとオペランド）の使用を許す。

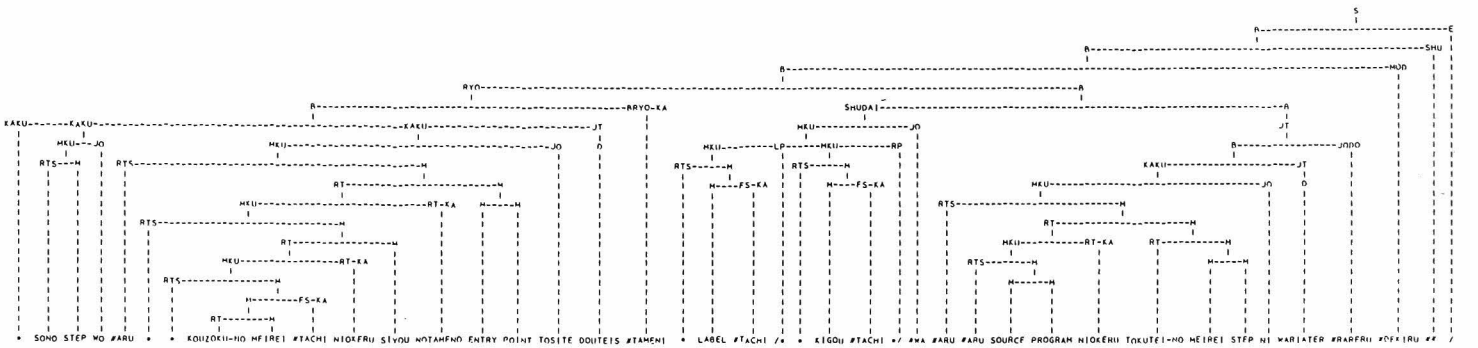
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((((/**/ (A* (*PL SYMBOL))) (A* (*PL LABEL)))
    (CAN
      (LAMBDA
        X456
        (((INF-AV
          (LAMBDA
            X447
            ((THAT** STEP)
              (LAMBDA
                X448
                ((A
                  (LAMBDA
                    X443
                    ((*AB
                      (LAMBDA
                        X441
                        ((A* (*PL (SUBSEQUENT INSTRUCTION)))
                          (LAMBDA X440 ((((*AP IN) X440) USE-N1) X441)))))
                      (LAMBDA
                        X442
                        ((((*AP FOR) X442) ((*ADJ-CLSF ENTRY) POINT)) X443)))))
                  (LAMBDA X449 (IDENTIFY X447 X448 X449)))))))
        (LAMBDA
          X452
          ((A
            (LAMBDA
              X451
              ((A ((*ADJ-CLSF SOURCE) PROGRAM))
                (LAMBDA
                  X450
                  ((((*AP IN) X450)
                    (PARTICULAR ((*ADJ-CLSF INSTRUCTION) STEP)))
                    X451)))))
            (LAMBDA X453 ((*EN3 ASSIGN-TO) X452 X453)))))
        X456)))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



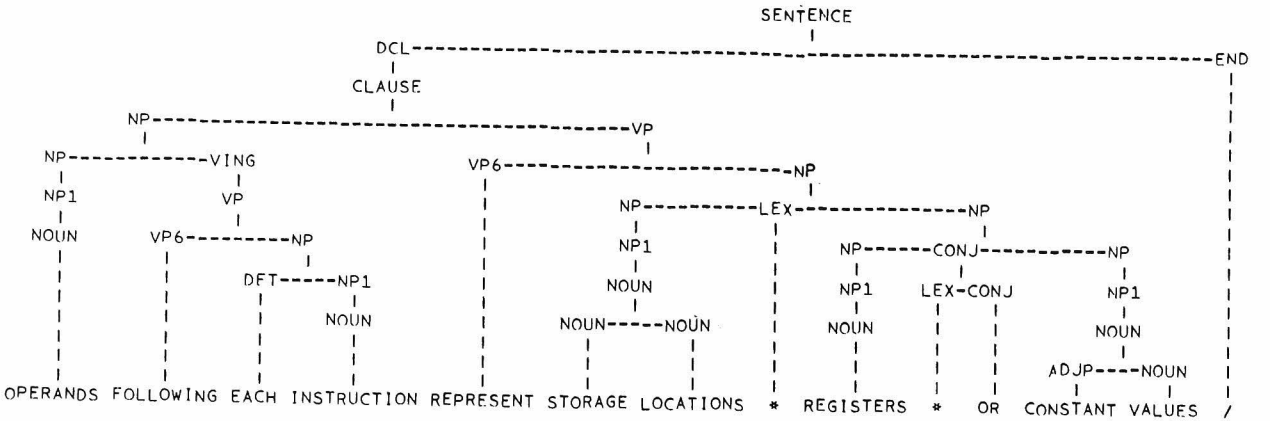
Input Sentence:

LABELS /* SYMBOLS */ CAN BE ASSIGNED TO A PARTICULAR INSTRUCTION STEP IN A SOURCE PROGRAM TO IDENTIFY THAT STEP AS AN ENTRY POINT FOR USE IN SUBSEQUENT INSTRUCTIONS /

Output Sentence:

ラベル（記号）はそのステップを後続の命令における使用のためのエントリーポイントとして同定するためにソースプログラムにおける特定の命令ステップに割り当てられることができる。

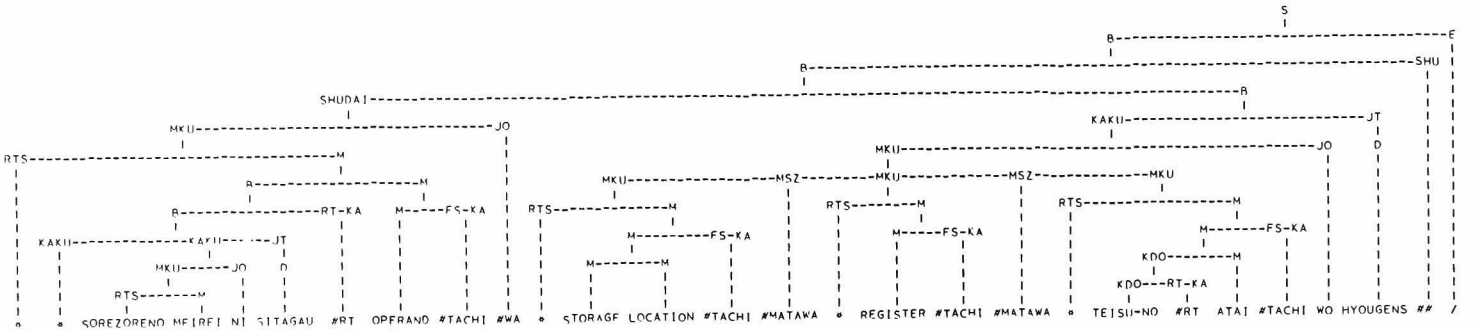
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((A*
    ((PART1
      (LAMBDA
        X275
        ((EACH INSTRUCTION) (LAMBDA X276 (FOLLOW1 X275 X276)))))
      (*PL OPERAND)))
    (LAMBDA
      X285
      ((*OR
        (A* (*PL ((*ADJ-CLSF STORAGE) LOCATION)))
        (A* (*PL REGISTER))
        (A* (*PL (CONSTANT-A VALUE))))
        (LAMBDA X286 (REPRESENT X285 X286))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



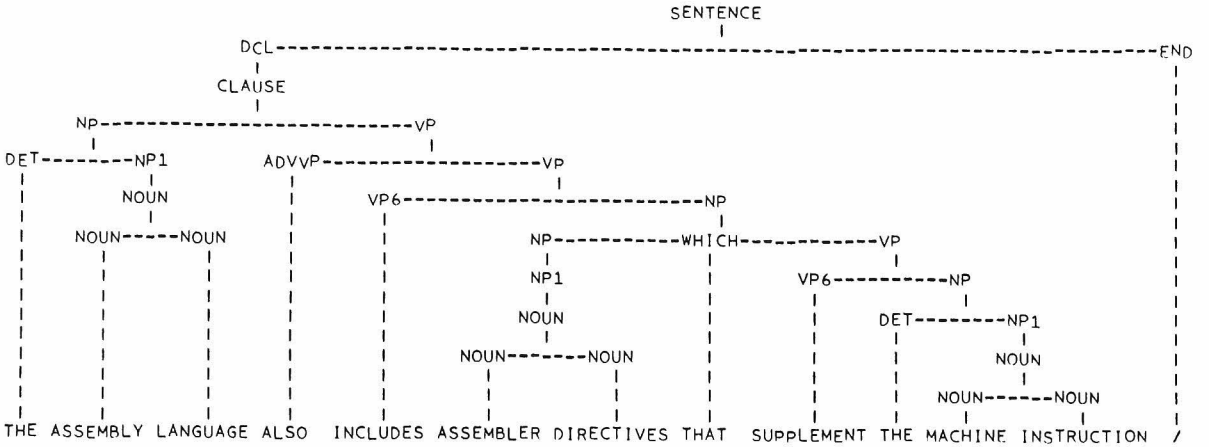
Input Sentence:

OPERANDS FOLLOWING EACH INSTRUCTION REPRESENT STORAGE LOCATIONS * REGISTERS * OR CONSTANT VALUES /

Output Sentence:

それぞれの命令に従うオペランドはストレージロケーションまたはレジスタまたは定数の値を表現する。

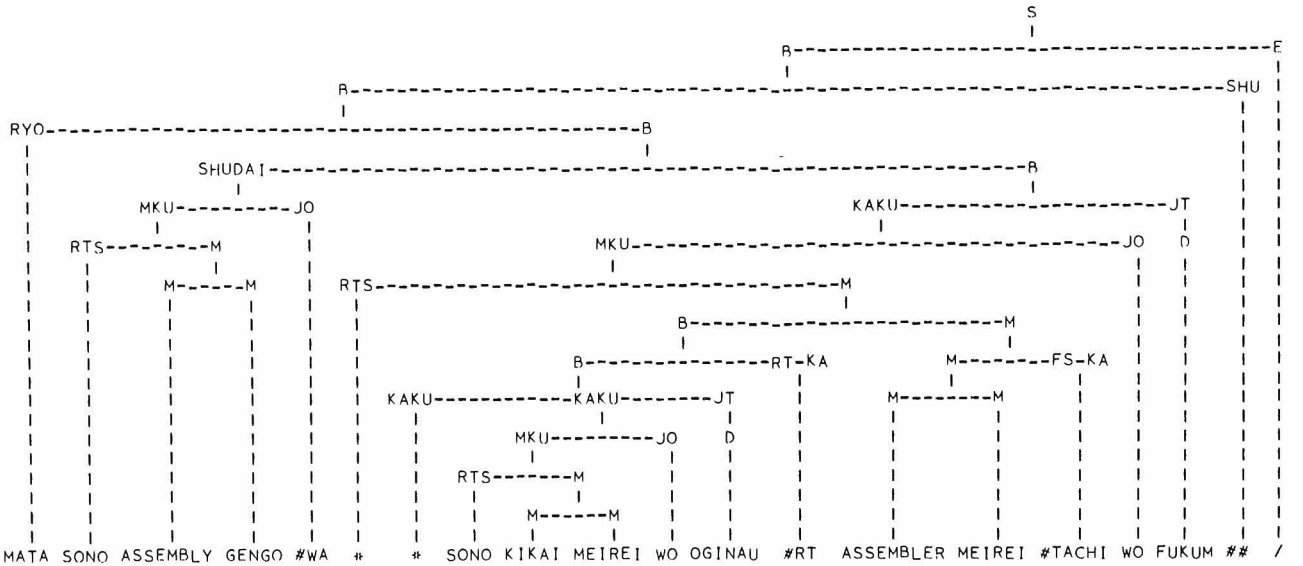
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((THE ((*ADJ-CLSF ASSEMBLY) LANGUAGE))
    (LAMBDA
      X307
      (ALSO
        ((A*
          ((WHICH
            (LAMBDA
              X291
              ((THE ((*ADJ-CLSF MACHINE) INSTRUCTION))
                (LAMBDA X292 (SUPPLEMENT-V X291 X292)))))
           (*PL ((*ADJ-CLSF ASSEMBLER) DIRECTIVE))))
         (LAMBDA X308 (INCLUDE X307 X308)))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



Input Sentence:

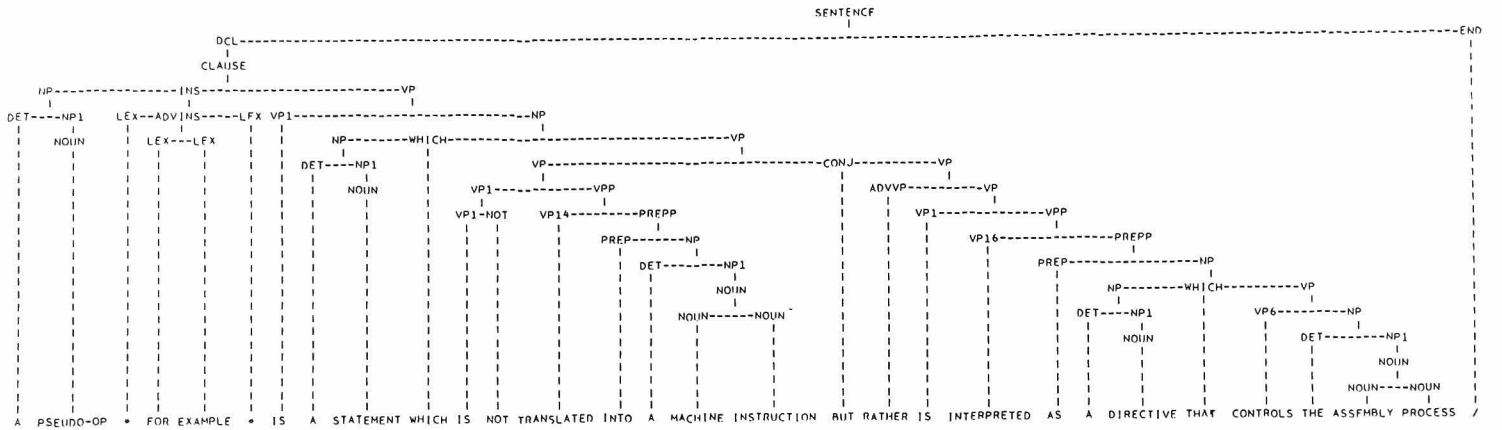
THE ASSEMBLY LANGUAGE ALSO INCLUDES ASSEMBLER DIRECTIVES THAT SUPPLEMENT THE MACHINE INSTRUCTION /

Output Sentence:

そのアセンブリ言語はまたその機械命令を補うアセンブラ命令を含む。

S-5

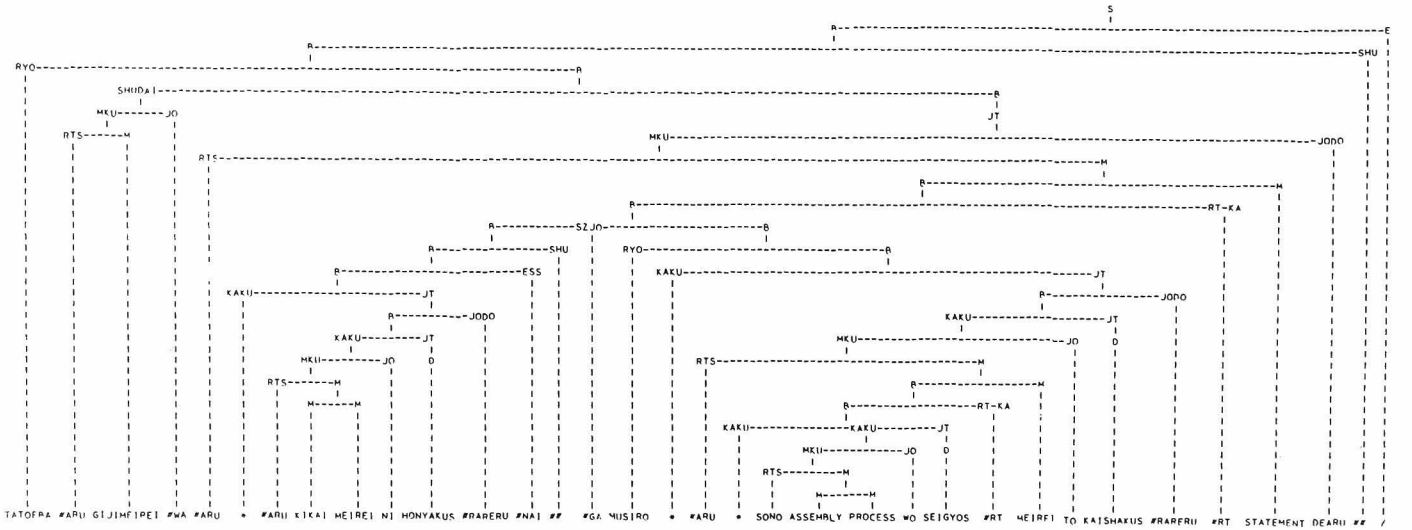
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (FOR-EXAMPLE-ADVINS
    ((A PSEUDO-OP)
      (LAMBDA
        X125
        ((A
          ((WHICH
            (BUT*VP
              (LAMBDA
                X122
                (NOT
                  ((A ((*ADJ-CLSF MACHINE) INSTRUCTION))
                    (LAMBDA X76 ((*EN3 TRANSLATE-INTO) X122 X76)))))
              (LAMBDA
                X123
                (RATHER-ADVVP
                  ((A
                    ((WHICH
                      (LAMBDA
                        X95
                        ((THE ((*ADJ-AP ASSEMBLY) PROCESS-N))
                          (LAMBDA X96 (CONTROL-V X95 X96)))))
                      DIRECTIVE))
                    (LAMBDA X118 ((*EN3 INTERPRET-AS) X123 X118)))))))
            STATEMENT))
        (LAMBDA X126 (IS X125 X126)))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



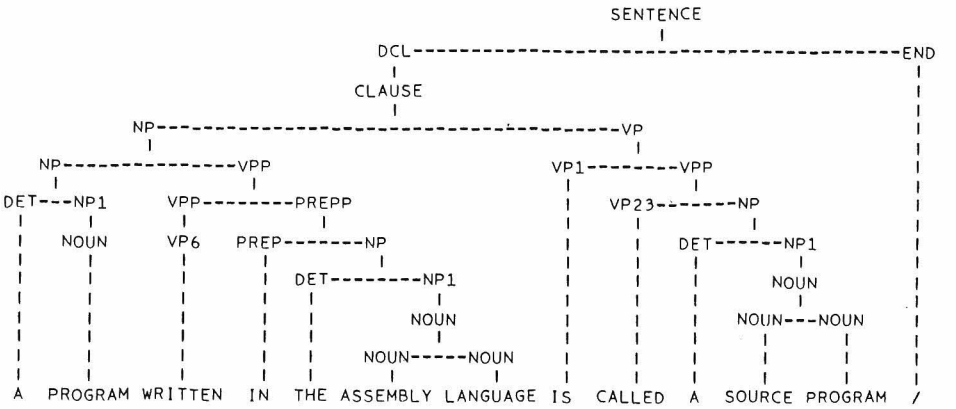
Input Sentence:

A PSEUDO-OP * FOR EXAMPLE * IS A STATEMENT WHICH IS NOT TRANSLATED INTO A MACHINE INSTRUCTION BUT RATHER IS INTERPRETED AS A DIRECTIVE THAT CONTROLS THE ASSEMBLY PROCESS /

Output Sentence:

擬似命令は例えば機械命令に翻訳されないが、むしろそのアセンブリ過程を制御する命令と解釈されるステートメントである。

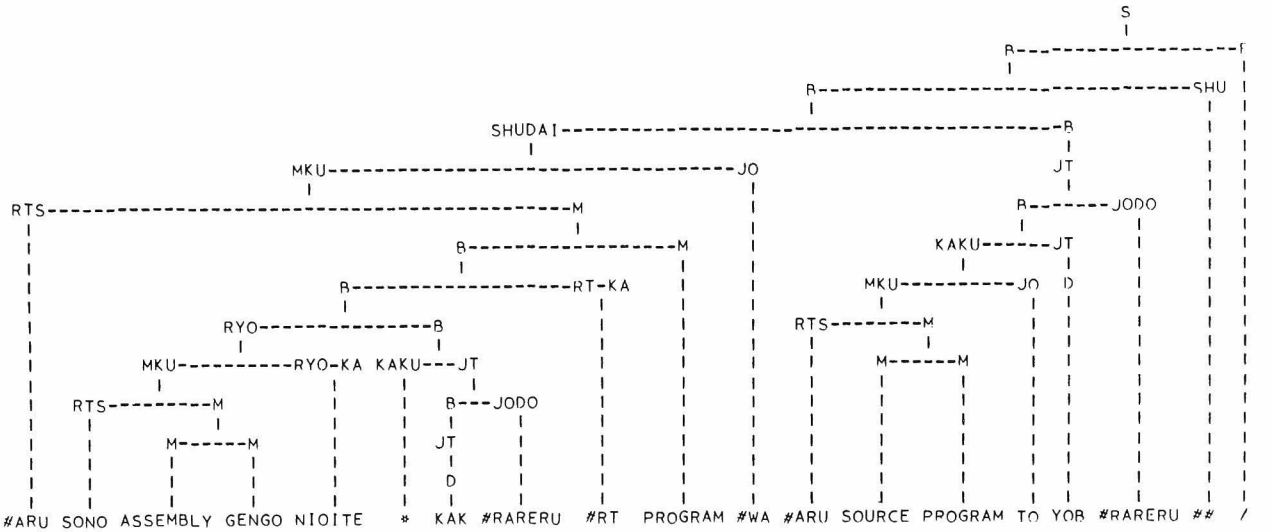
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((A
    ((PARTI
      (LAMBDA
        X200
        ((THE ((*ADJ ASSEMBLY) LANGUAGE))
          (LAMBDA X199 ((IN13 X199) ((*EN WRITE) X200))))))
      PROGRAM))
    (LAMBDA
      X203
      ((A ((*ADJ SOURCE) PROGRAM))
        (LAMBDA X204 ((*EN3 CALL-V3) X203 X204))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



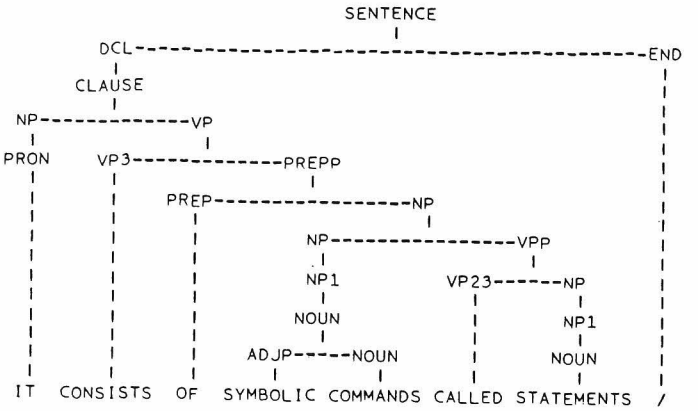
Input Sentence:

A PROGRAM WRITTEN IN THE ASSEMBLY LANGUAGE IS CALLED A SOURCE PROGRAM /

Output Sentence:

そのアセンブリ言語において書かれるプログラムはソースプログラムと呼ばれる。

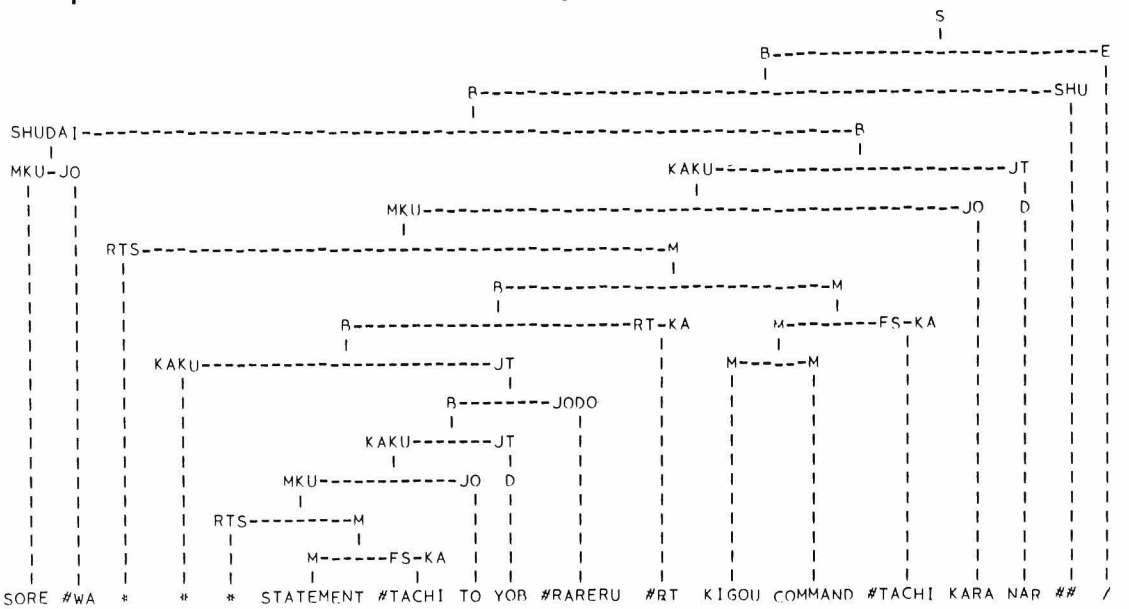
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (IT
    (LAMBDA
      X112
      ((A*
        ((PARTI
          (LAMBDA
            X110
            ((A* (*PL STATEMENT))
              (LAMBDA X111 ((*EN3 CALL-V3) X110 X111)))))
          (*PL (SYMBOLIC COMMAND))))
        (LAMBDA X113 (CONSIST-OF X112 X113))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



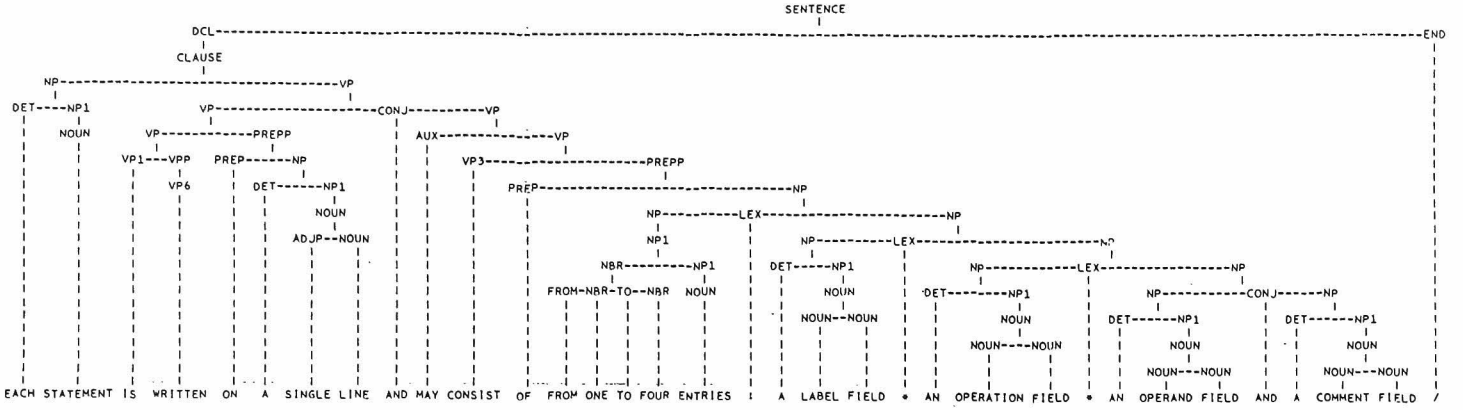
Input Sentence:

IT CONSISTS OF SYMBOLIC COMMANDS CALLED STATEMENTS /

Output Sentence:

それはステートメントと呼ばれる記号コマンドから成る。

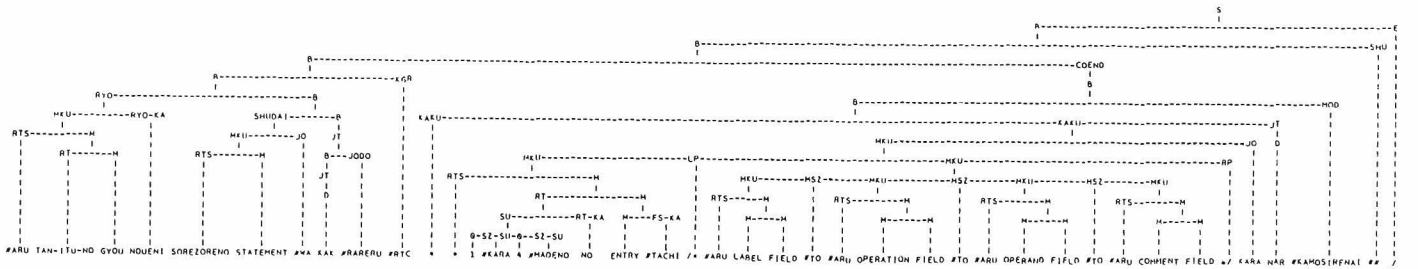
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((EACH STATEMENT)
    (AND*VP
      (LAMBDA
        X185
        ((A (SINGLE LINE))
          (LAMBDA X184 ((ON X184) ((*EN WRITE) X185)))))
      (MAY
        (LAMBDA
          X208
          (((:
            (*AND
              (A ((*ADJ-CLSF LABEL) FIELD))
              (A ((*ADJ-CLSF OPERATION) FIELD))
              (A ((*ADJ-CLSF OPERAND) FIELD))
              (A ((*ADJ-CLSF COMMENT) FIELD))))
            (A* ((FROM#TO# (QUOTE 1) (QUOTE 4)) (*PL ENTRY))))
            (LAMBDA X209 (CONSIST-OF X208 X209))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



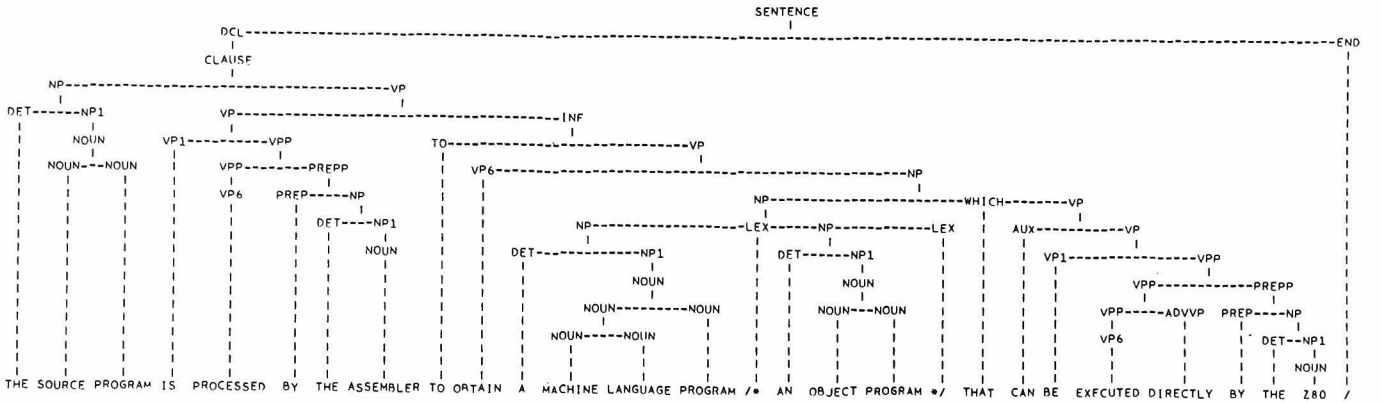
Input Sentence:

EACH STATEMENT IS WRITTEN ON A SINGLE LINE AND MAY CONSIST OF FROM ONE TO FOUR ENTRIES : A LABEL FIELD * AN OPERATION FIELD * AN OPERAND FIELD AND A COMMENT FIELD /

Output Sentence:

それぞれのステートメントは単一の行の上に書かれ，1から4までのエントリー（ラベルフィールドと操作フィールドとオペランドフィールドとコメントフィールド）から成るかもしれない。

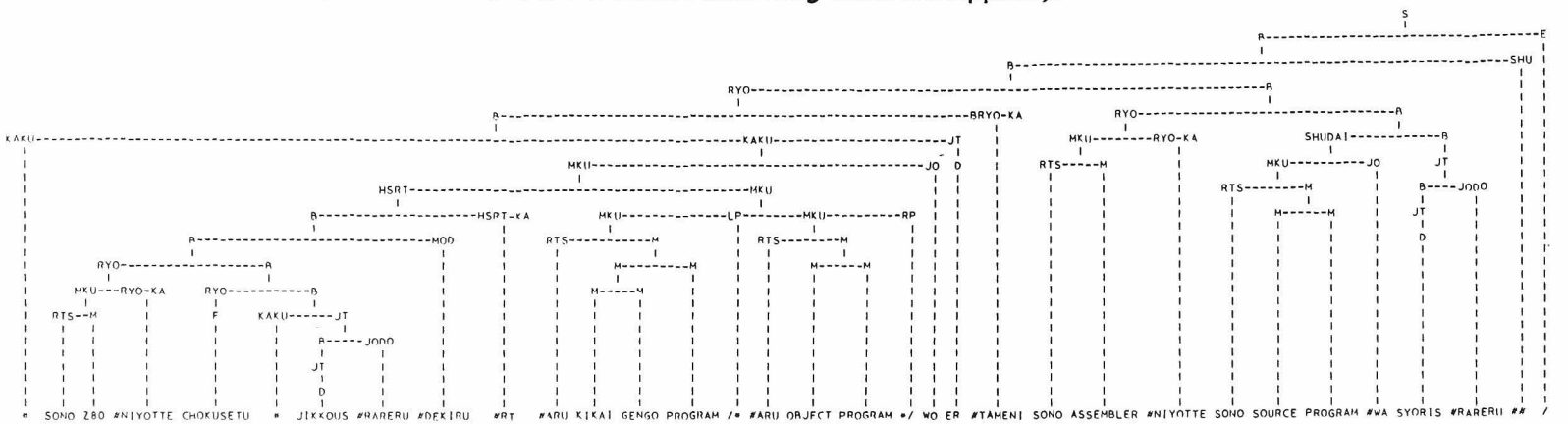
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((THE ((*ADJ-CLSF SOURCE) PROGRAM))
    (LAMBDA
      X377
      (((INF-AV
        (LAMBDA
          X241
          (((WHICH-NR
            (CAN
              (LAMBDA
                X220
                ((THE Z80)
                  (LAMBDA
                    X219
                    ((*PSUBJ X219) (DIRECTLY ((*EN EXECUTE) X220))))))))
            ((/**/ (A ((*ADJ-CLSF OBJECT) PROGRAM)))
              (A ((*ADJ-CLSF ((*ADJ-CLSF MACHINE) LANGUAGE)) PROGRAM))))
            (LAMBDA X242 (OBTAIN X241 X242)))))
        (LAMBDA
          X370
          ((THE ASSEMBLER)
            (LAMBDA X369 ((*PSUBJ X369) ((*EN PROCESS-V) X370))))))
        X377))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



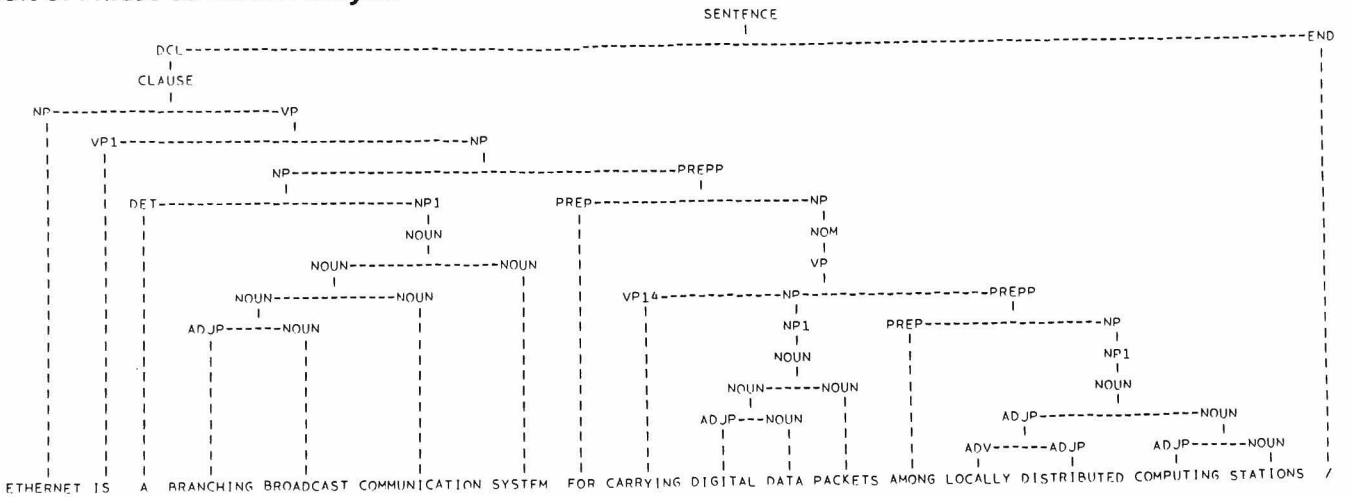
Input Sentence:

THE SOURCE PROGRAM IS PROCESSED BY THE ASSEMBLER TO OBTAIN A MACHINE LANGUAGE PROGRAM /* AN OBJECT PROGRAM */ THAT CAN BE EXECUTED DIRECTLY BY THE Z80 /

Output Sentence:

そのソースプログラムはそのＺ８０によって直接実行されることができる機械言語プログラム（オブジェクトプログラム）を得るためにそのアセンブラによって処理される。

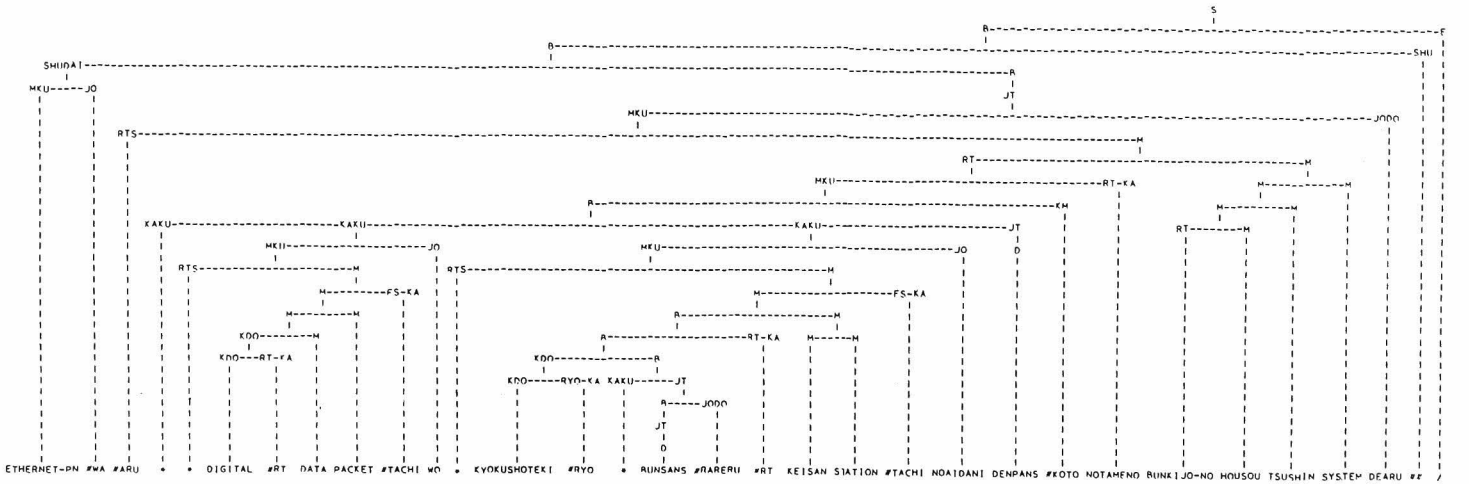
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (ETHERNET-PN
    (LAMBDA
      X21
      ((A
        (LAMBDA
          X20
          ((NOM
            (LAMBDA
              X14
              ((A* (*PL ((*ADJ-CLSF (DIGITAL DATA)) PACKET)))
                (LAMBDA
                  X15
                  ((A*
                    (*PL
                      ((PART1 (LOCALLY (*EN DISTRIBUTE)))
                        (COMPUTING STATION))))
                    (LAMBDA X16 (CARRY-AMONG X14 X15 X16)))))))
            (LAMBDA
              X19
              ((((*AP FOR) X19)
                ((*ADJ-CLSF
                  ((*ADJ-CLSF (BRANCHING BROADCAST)) COMMUNICATION))
                  SYSTEM))
                X20)))))
        (LAMBDA X22 (IS X21 X22))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



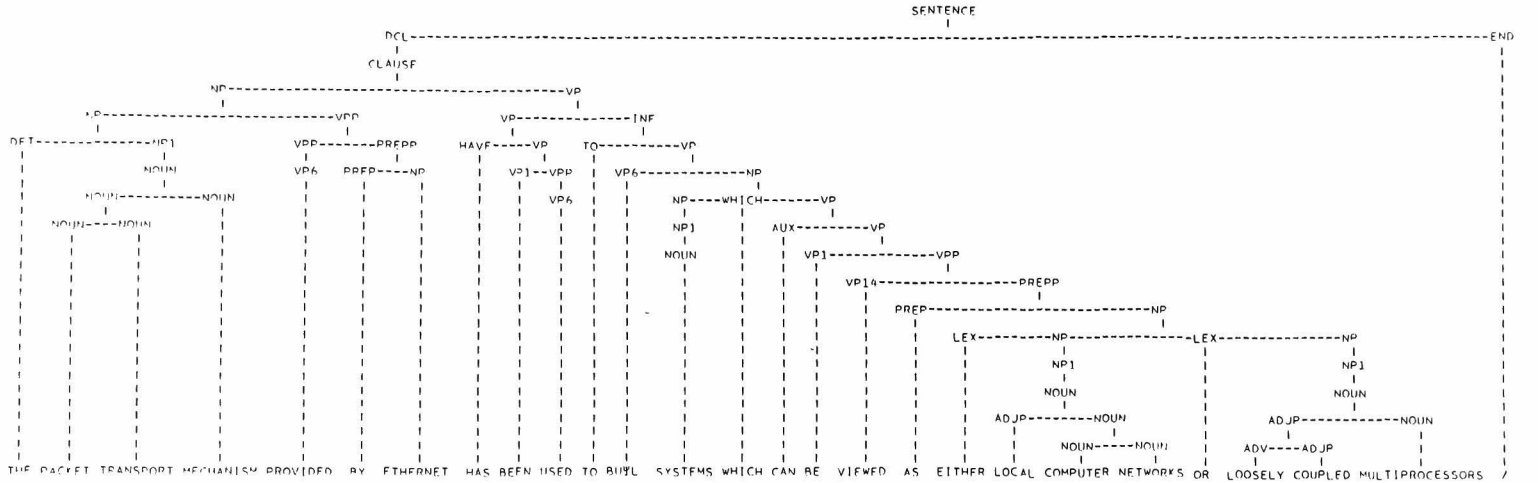
Input Sentence:

ETHERNET IS A BRANCHING BROADCAST COMMUNICATION SYSTEM FOR CARRYING DIGITAL DATA PACKETS AMONG LOCALLY DISTRIBUTED COMPUTING STATIONS /

Output Sentence:

ETHERNETはデジタルデータのパケットを局所的に分散される計算ステーションの間に伝搬するための分岐状の放送の通信のシステムである。

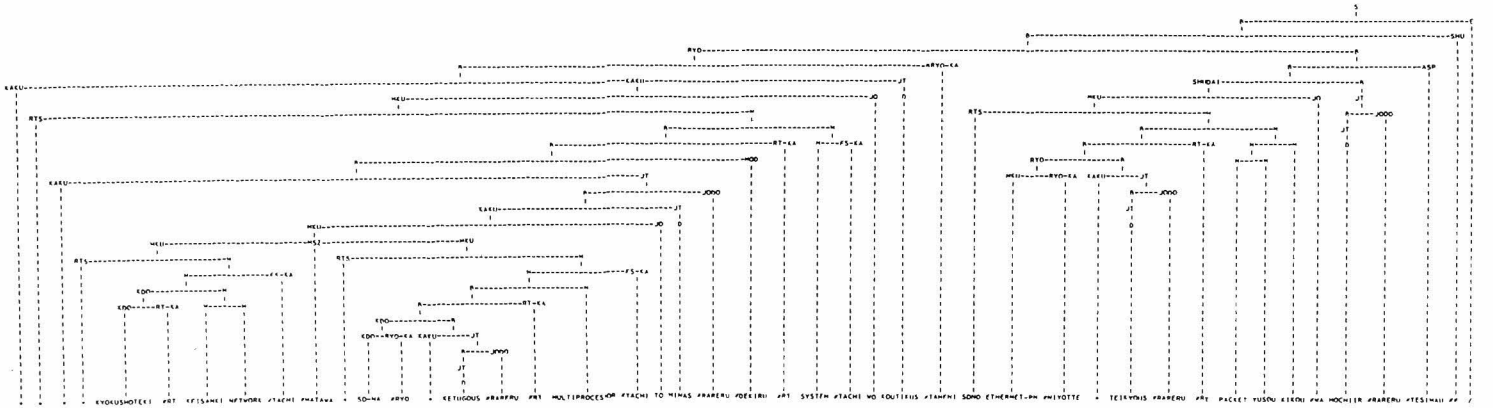
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((THE
    ((PARTI
      (LAMBDA
        X9
        (ETHERNET-PN (LAMBDA X8 ((*PSUBJ X8) ((*EN PROVIDE) X9))))))
      ((*ADJ-CLSF ((*ADJ-CLSF PACKET) TRANSPORT)) MECHANISM)))
    (LAMBDA
      X18
      (((INF-AV
        (LAMBDA
          X16
          ((A*
            ((WHICH
              (CAN
                (LAMBDA
                  X14
                  ((EITHER-OR-NP
                    (A* (*PL (LOCAL ((*ADJ-CLSF COMPUTER) NETWORK))))
                    (A*
                      (*PL
                        ((PARTI (LOOSELY (*EN COUPLE-V6))) MULTIPROCESSOR))))
                    (LAMBDA X15 ((*EN3 VIEW-AS) X14 X15))))))
              (*PL SYSTEM)))
            (LAMBDA X17 (BUILD-1 X16 X17)))))
        (#PERF (LAMBDA X10 ((*EN USE-V1) X10))))
        X18))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



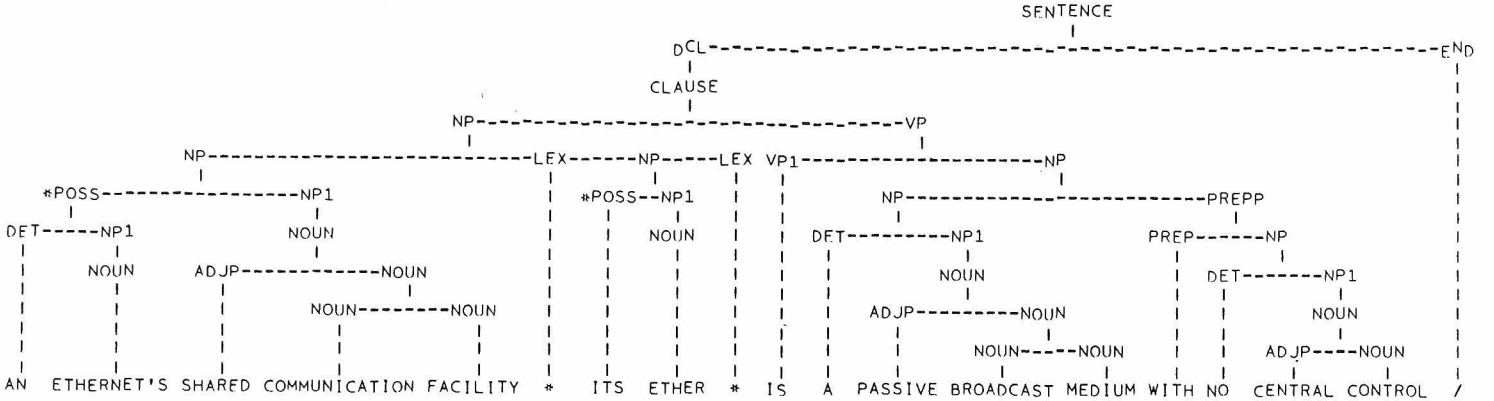
Input Sentence:

THE PACKET TRANSPORT MECHANISM PROVIDED BY ETHERNET HAS BEEN USED TO BUILD SYSTEMS WHICH CAN BE VIEWED AS EITHER LOCAL COMPUTER NETWORKS OR LOOSELY COUPLED MULTIPROCESSORS /

Output Sentence:

局所的な計算機ネットワークまたは疎に結合されるマルチプロセッサとみなされることができるシステムを構築するためにそのETHERNETによって提供されるパケット輸送機構は用いられてしまう。

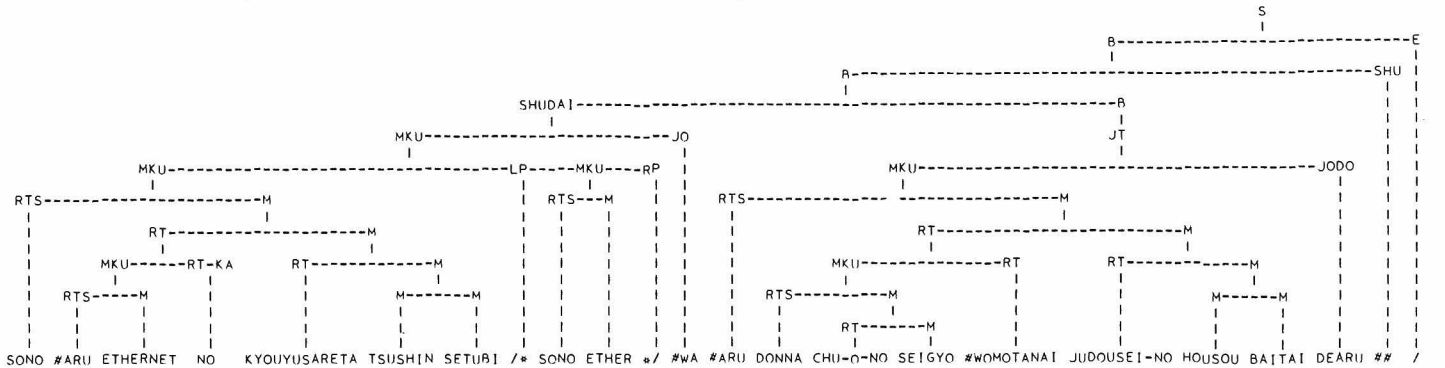
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (((** (ITS ETHER))
    (LAMBDA
      X27
      ((A ETHERNET)
        (LAMBDA
          X25
          (((*POSS X25) (SHARED ((*ADJ-CLSF COMMUNICATION) FACILITY)))
            X27)))))
    (LAMBDA
      X38
      ((A
        (LAMBDA
          X37
          ((*NO (CENTRAL CONTROL-N))
            (LAMBDA
              X36
              ((((*AP WITH1) X36)
                (PASSIVE ((*ADJ-CLSF BROADCAST) MEDIUM)))
                X37)))))
        (LAMBDA X39 (IS X38 X39))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



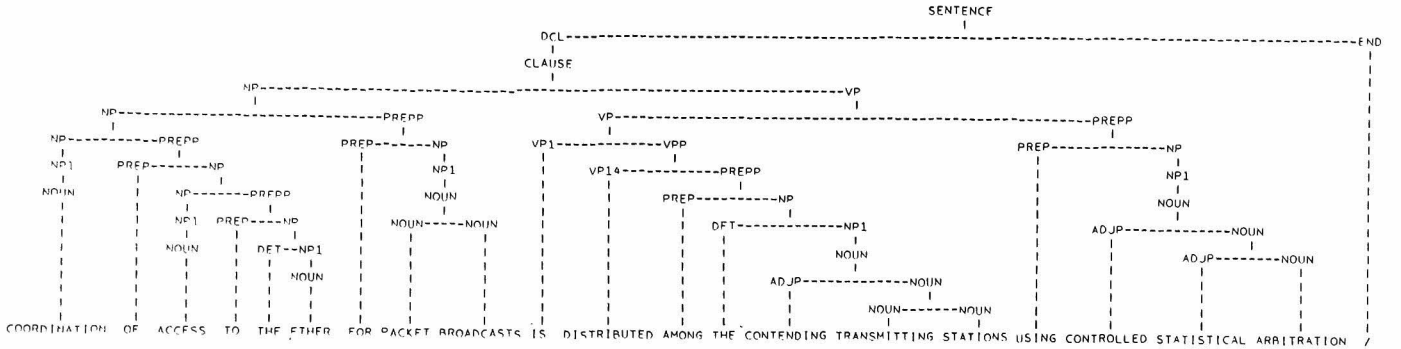
Input Sentence:

AN ETHERNET'S SHARED COMMUNICATION FACILITY * ITS ETHER * IS A PASSIVE BROADCAST MEDIUM WITH NO CENTRAL CONTROL /

Output Sentence:

ETHERNETの共有された通信設備（そのETHER）は中央の制御を持たない受動性の放送媒体である。

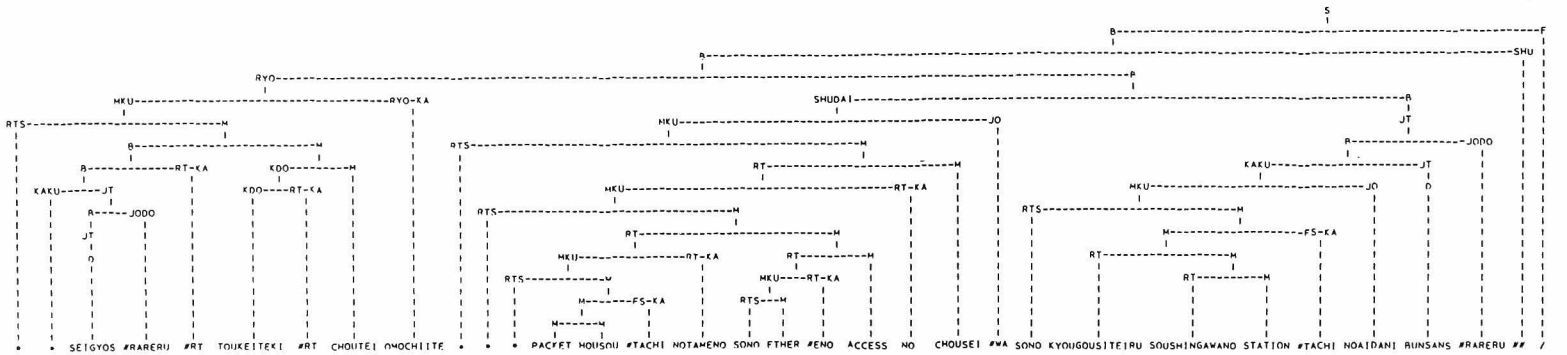
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((*AB
    (LAMBDA
      X67
      ((A* (*PL ((*ADJ PACKET) BROADCAST)))
        (LAMBDA
          X66
          ((((*AP FOR) X66)
            (LAMBDA
              X65
              ((*AB
                (LAMBDA
                  X63
                  ((THE ETHER)
                    (LAMBDA X62 ((((*AP TO) X62) ACCESS) X63)))))
                (LAMBDA X64 ((((*AP OF) X64) COORDINATION) X65)))))
           X67)))))
   (LAMBDA
     X93
     ((*AB (CONTROLLED (STATISTICAL ARBITRATION)))
       (LAMBDA
         X92
         ((USING X92)
           ((THE (*PL (CONTENDING ((*ADJ TRANSMITTING) STATION))))
             (LAMBDA X91 ((*EN3 DISTRIBUTE-AMONG) X93 X91)))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



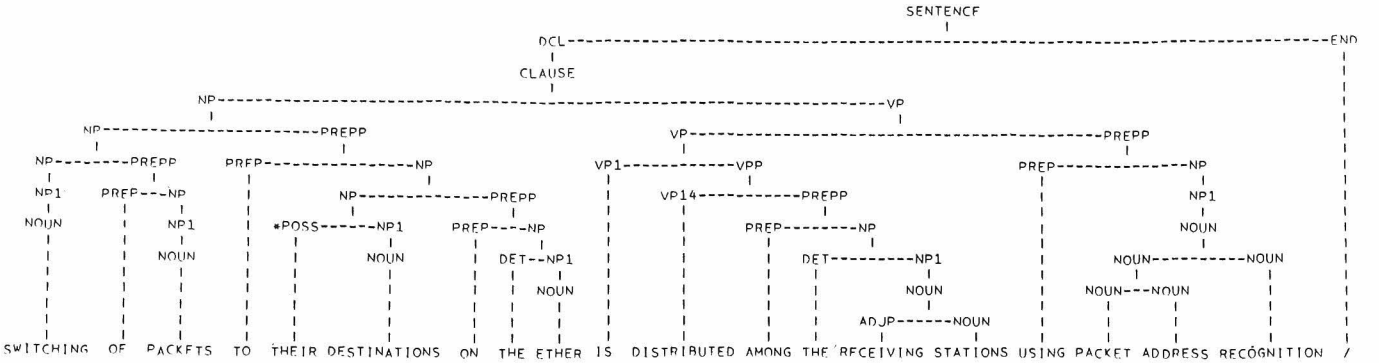
Input Sentence:

COORDINATION OF ACCESS TO THE ETHER FOR PACKET BROADCASTS IS DISTRIBUTED AMONG THE CONTENDING TRANSMITTING STATIONS USING CONTROLLED STATISTICAL ARBITRATION /

Output Sentence:

パケット放送のためのそのETHERへのアクセスの調整は制御される統計的調停を用いて競合している送信側のステーションの間に分散される。

Result of Phrase Structure Analysis:

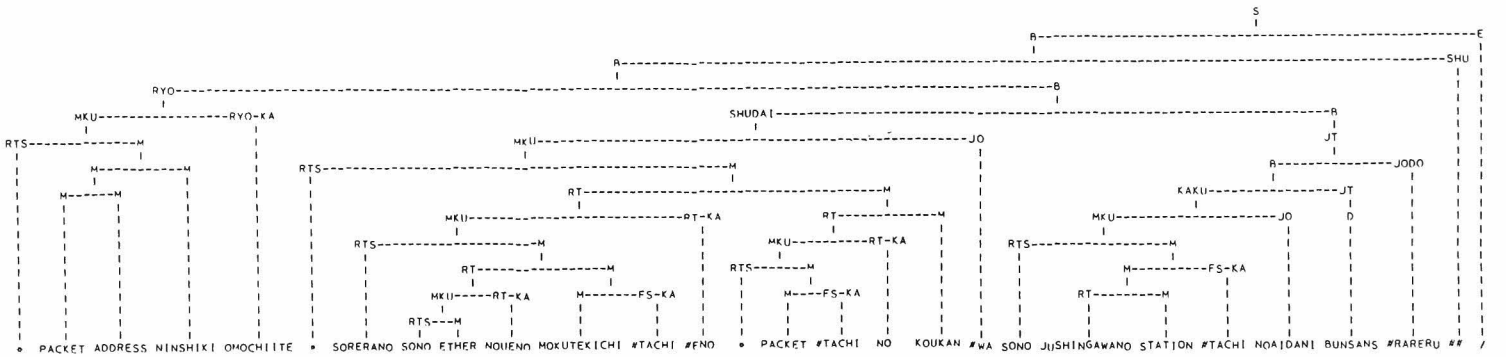


Extracted EFR Expression:

```
(#DCL
  ((*AB
    (LAMBDA
      X64
      (THEY
        (LAMBDA
          X50
          ((((*POSS X50)
            (LAMBDA
              X62
              ((THE ETHER)
                (LAMBDA X61 ((((*AP ON) X61) (*PL DESTINATION)) X62)))))

            (LAMBDA
              X63
              (((((*AP SWITCH-TO) X63)
                (LAMBDA
                  X49
                  ((A* (*PL PACKET))
                    (LAMBDA X48 ((((*AP OF) X48) SWITCHING) X49)))))
                X64)))))))
   (LAMBDA
     X60
     ((*AB ((*ADJ-OBJ ((*ADJ-CLSF PACKET) ADDRESS)) RECOGNITION))
       (LAMBDA
         X59
         ((USING X59)
           ((THE (*PL (RECEIVING STATION)))
             (LAMBDA X58 ((*EN3 DISTRIBUTE-AMONG) X60 X58)))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



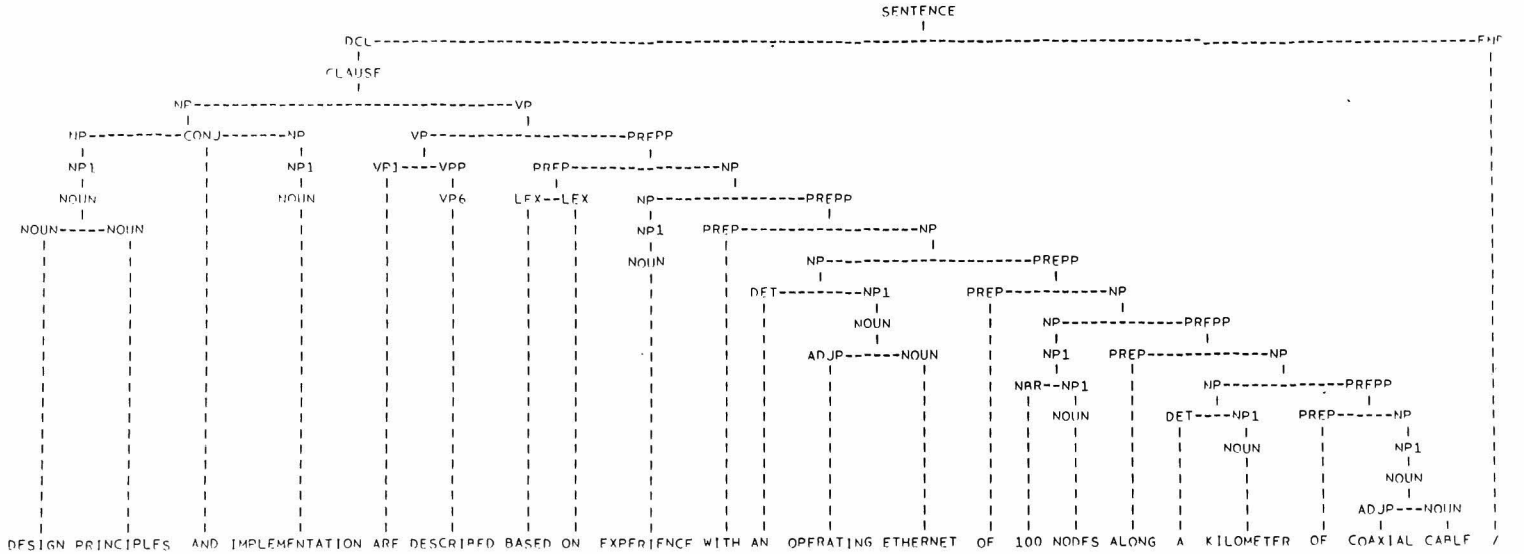
Input Sentence:

SWITCHING OF PACKETS TO THEIR DESTINATIONS ON THE ETHER IS DISTRIBUTED AMONG THE RECEIVING STATIONS USING PACKET ADDRESS RECOGNITION /

Output Sentence:

そのETHERの上の目的地へのパケットの交換はパケット番地認識を用いて受信側のステーションの間に分散される。

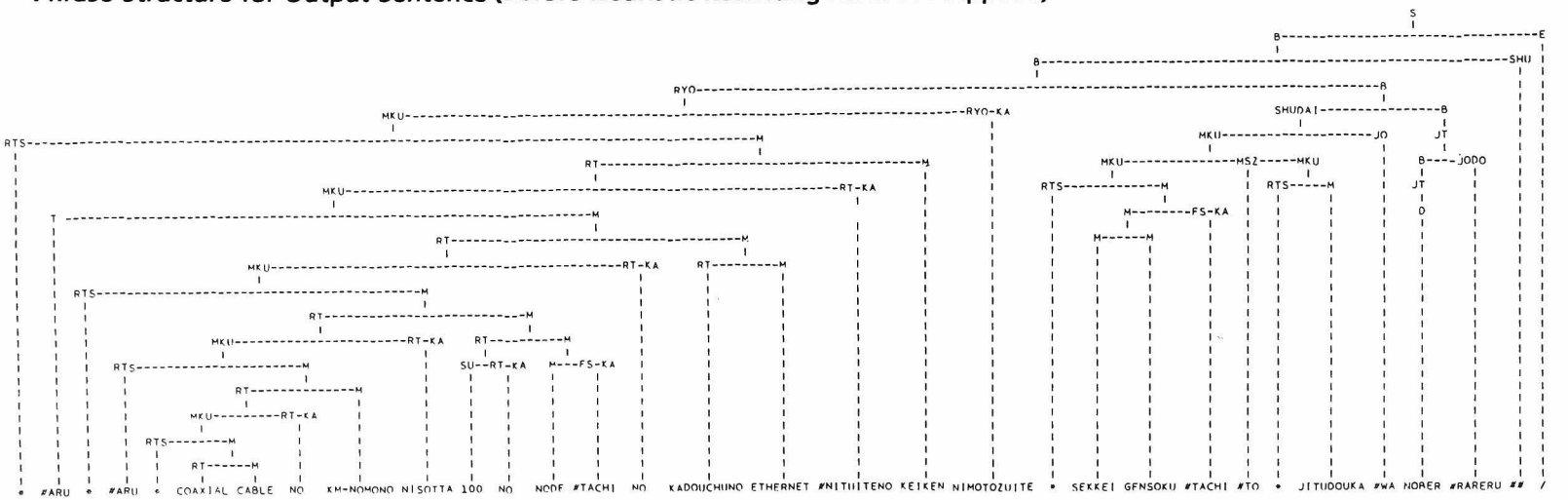
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((*AND
    (A* (*PL ((*ADJ-CLSF DESIGN-N) PRINCIPLE)))
    (*AR IMPLEMENTATION))
    (LAMBDA
      X420
      ((*AB
        (LAMBDA
          X418
          ((A
            (LAMBDA
              X400
              ((A*
                (LAMBDA
                  X384
                  ((A
                    (LAMBDA
                      X382
                      ((*AB (COAXIAL CABLE))
                        (LAMBDA X381 ((((*AP OF) X381) KILOMETER) X382)))))
                    (LAMBDA
                      X383
                      ((((*AP ALONG) X383) ((*NBR (QUOTE 100)) (*PL NODE)))
                        X384)))))
                (LAMBDA
                  X399
                  ((((*AP OF) X399) (OPERATING ETHERNET)) X400)))))
            (LAMBDA X417 ((((*AP WITH) X417) EXPERIENCE) X418)))))
      (LAMBDA X419 ((BASED-ON X419) ((*EN DESCRIBE) X420)))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



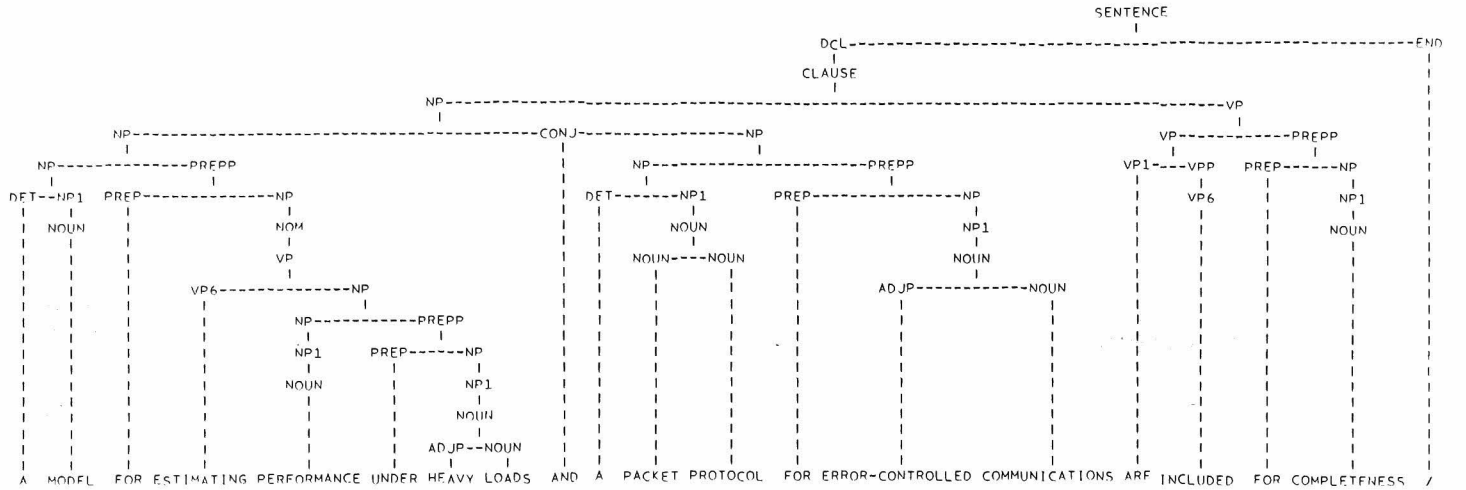
Input Sentence:

DESIGN PRINCIPLES AND IMPLEMENTATION ARE DESCRIBED BASED ON EXPERIENCE WITH AN OPERATING ETHERNET OF 100 NODES ALONG A KILOMETER OF COAXIAL CABLE /

Output Sentence:

設計原則と実働化は１Kmのコアキシャルケーブルに沿った１００のノードの稼動中のETHERNETについての経験に基づいて述べられる。

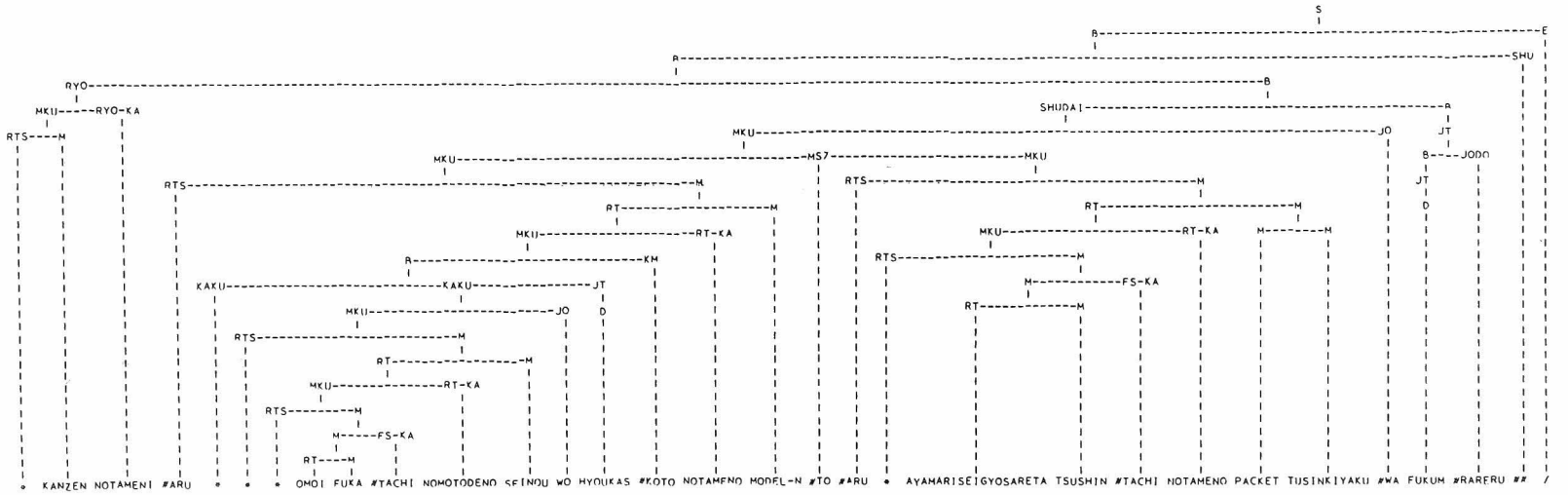
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((*AND
    (A
      (LAMBDA
        X64
        ((NOM
          (LAMBDA
            X13
            ((*AB
              (LAMBDA
                X12
                ((A* (*PL (HEAVY LOAD)))
                  (LAMBDA X11 ((((*AP UNDER) X11) PERFORMANCE) X12)))))
              (LAMBDA X14 (ESTIMATE X13 X14)))))
          (LAMBDA X63 ((((*AP FOR) X63) MODEL-N) X64)))))
    (A
      (LAMBDA
        X9
        ((A* (*PL (ERROR-CONTROLLED COMMUNICATION)))
          (LAMBDA
            X8
            ((((*AP FOR) X8) ((*ADJ-CLSF PACKET) PROTOCOL)) X9)))))))
    (LAMBDA
      X71
      ((*AB COMPLETENESS)
        (LAMBDA X70 ((FOR X70) ((*EN INCLUDE) X71)))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



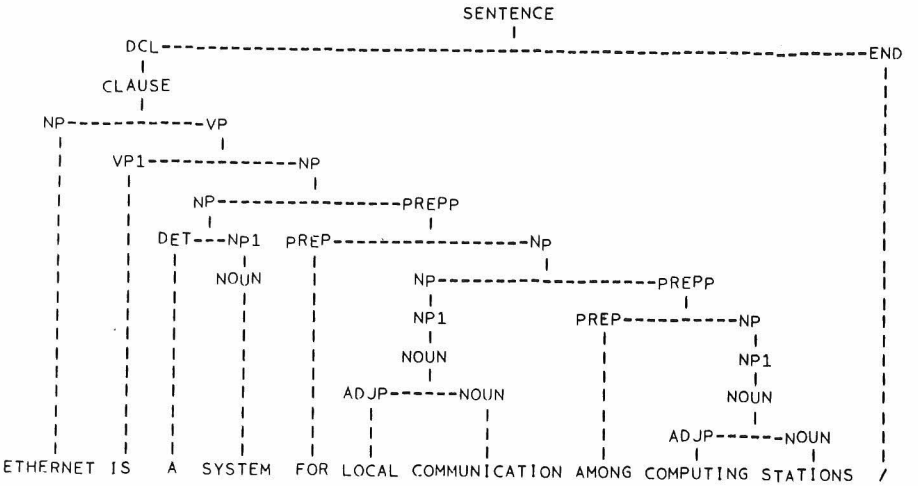
Input Sentence:

A MODEL FOR ESTIMATING PERFORMANCE UNDER HEAVY LOADS AND A PACKET PROTOCOL FOR ERROR-CONTROLLED COMMUNICATIONS ARE INCLUDED FOR COMPLETENESS /

Output Sentence:

重い負荷の下での性能を評価するためのモデルと誤り制御された通信のためのパケット通信規約は完全のために含まれる。

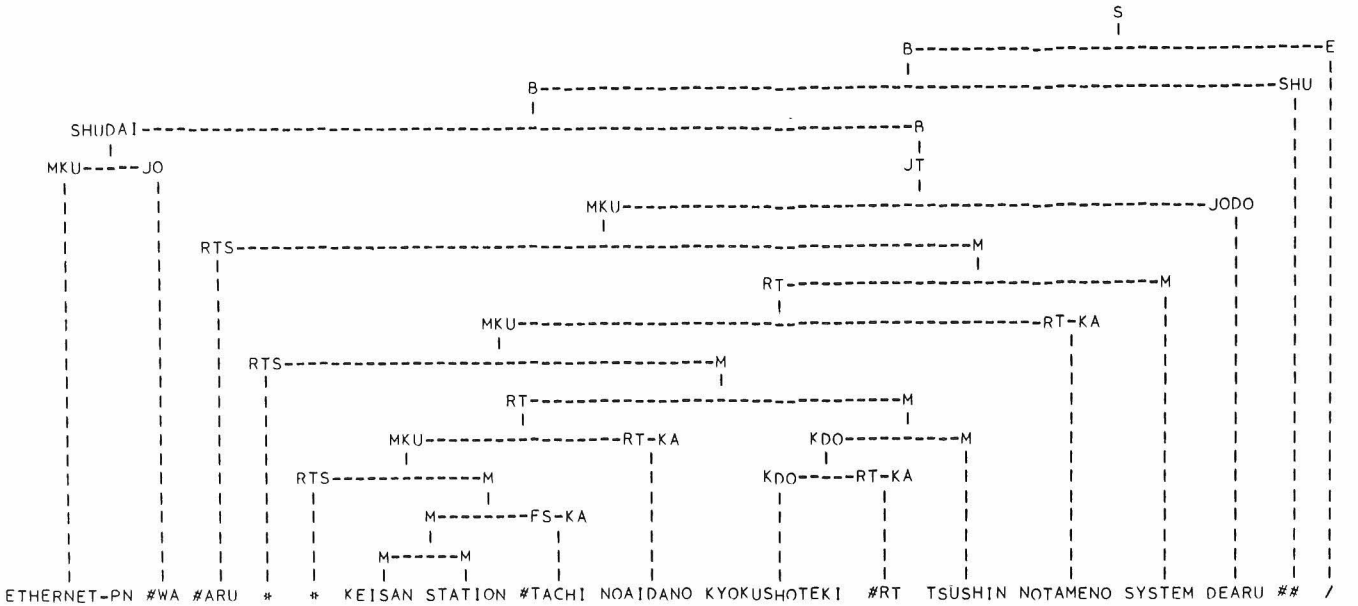
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (ETHERNET-PN
    (LAMBDA
      X16
      ((A
        (LAMBDA
          X15
          ((*AB
            (LAMBDA
              X7
              ((A* (*PL (COMPUTING STATION)))
                (LAMBDA
                  X6
                  ((((*AP AMONG) X6) (LOCAL COMMUNICATION))
                    X7)))))
            (LAMBDA
              X14
              ((((*AP FOR1) X14) SYSTEM) X15))))))
       (LAMBDA X17 (IS X16 X17))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



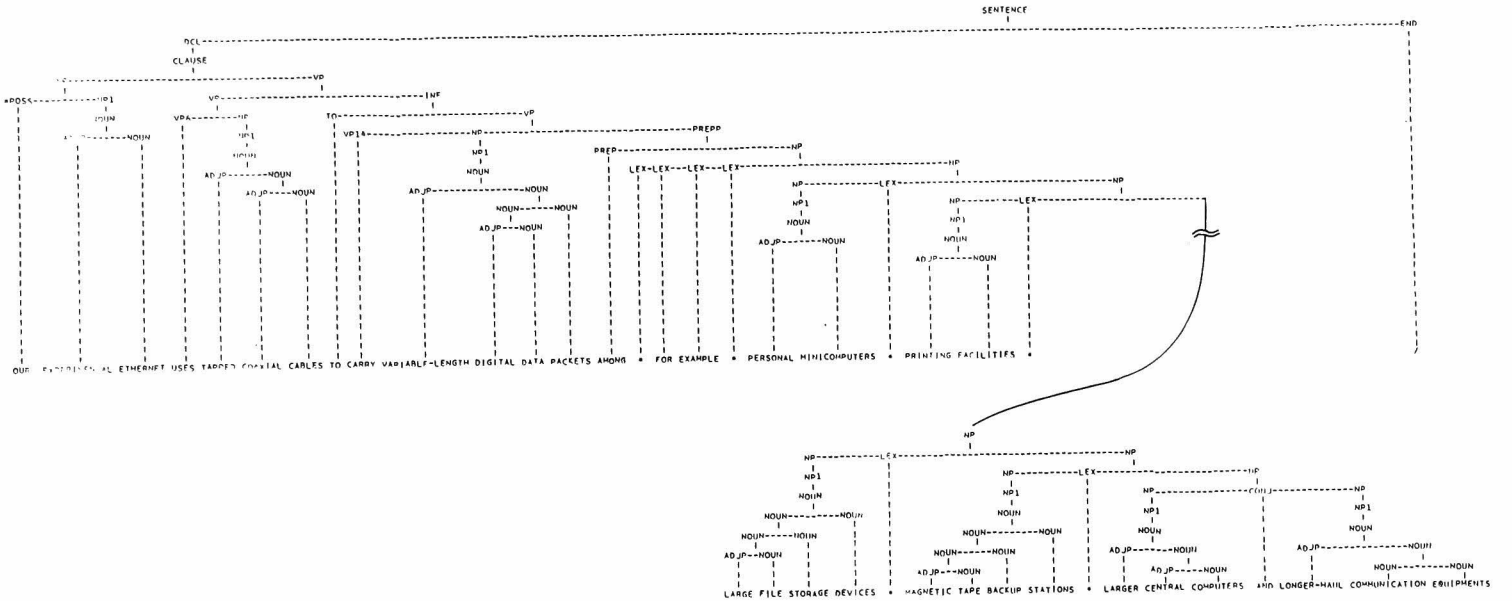
Input Sentence:

ETHERNET IS A SYSTEM FOR LOCAL COMMUNICATION AMONG COMPUTING STATIONS /

Output Sentence:

ETHERNETは計算ステーションの間の局所的な通信のためのシステムである。

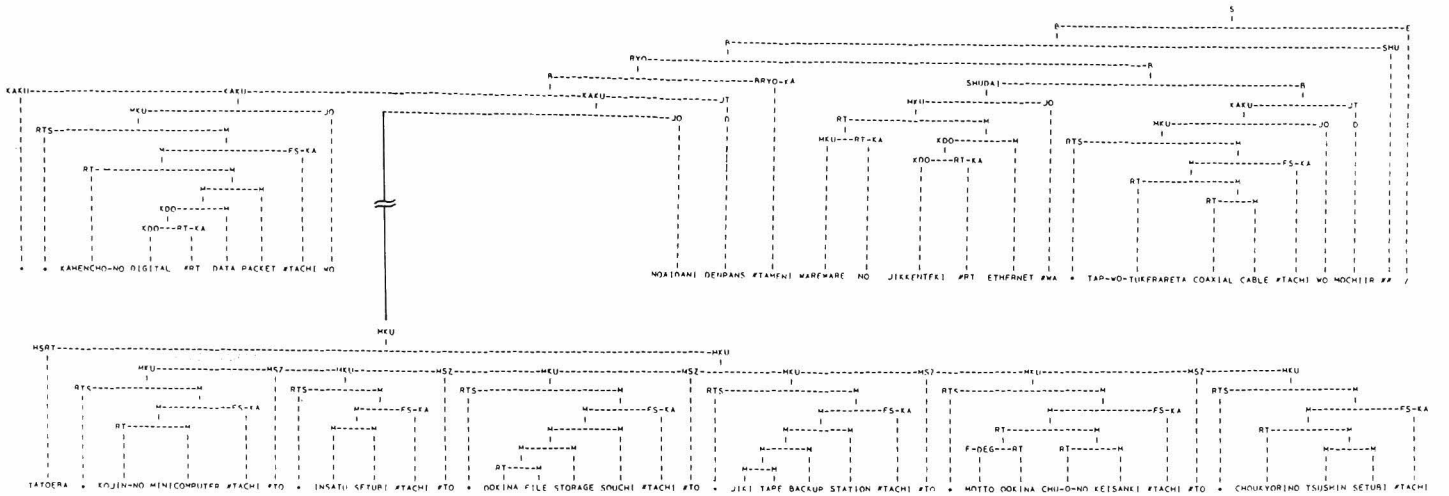
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
 ((OUR (EXPERIMENTAL ETHERNET))
   (LAMBDA
     X259
     ((((INF-AV
       (LAMBDA
         X256
         ((A*
           (*PL
             (VARIABLE-LENGTH ((*ADJ-CLSF (DIGITAL DATA)) PACKET))))
           (LAMBDA
             X257
             ((EG*NP
               (*AND
                 (A* (*PL (PERSONAL MINICOMPUTER)))
                 (A* (*PL (PRINTING FACILITY)))
                 (A*
                   (*PL
                     ((*ADJ ((*ADJ-OBJ (LARGE FILE)) STORAGE)) DEVICE)))
                 (A*
                   (*PL
                     ((*ADJ-CLSF ((*ADJ-OBJ (MAGNETIC TAPE)) BACKUP))
                       STATION)))
                 (A* (*PL ((MORE LARGE) (CENTRAL COMPUTER))))
                 (A*
                   (*PL
                     (LONGER-HAUL ((*ADJ-CLSF COMMUNICATION) EQUIPMENT))))))
              (LAMBDA X258 (CARRY-AMONG X256 X257 X258)))))))
      (LAMBDA
        X134
        ((A* (*PL (TAPPED (COAXIAL CABLE))))
          (LAMBDA X135 (USE-V1 X134 X135)))))
     X259))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



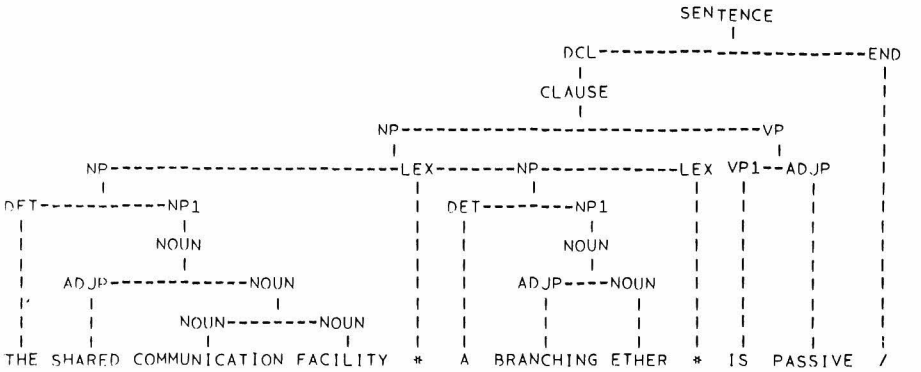
Input Sentence:

OUR EXPERIMENTAL ETHERNET USES TAPPED COAXIAL CABLES TO CARRY VARIABLE-LENGTH DIGITAL DATA PACKETS AMONG * FOR EXAMPLE * PERSONAL MINICOMPUTERS * PRINTING FACILITIES * LARGE FILE STORAGE DEVICES * MAGNETIC TAPE BACKUP STATIONS * LARGER CENTRAL COMPUTERS * AND LONGER-HAUL COMMUNICATION EQUIPMENTS /

Output Sentence:

我々の実験的なETHERNETは可変長のデジタルデータのパケットを例えば個人のミニコンピュータと印刷設備と大きなファイルのストレージの装置と磁気テープバックアップステーションともっと大きな中央の計算機と長距離の通信設備の間に伝搬するためにタップを付けられたコアキシャルケーブルを用いる。

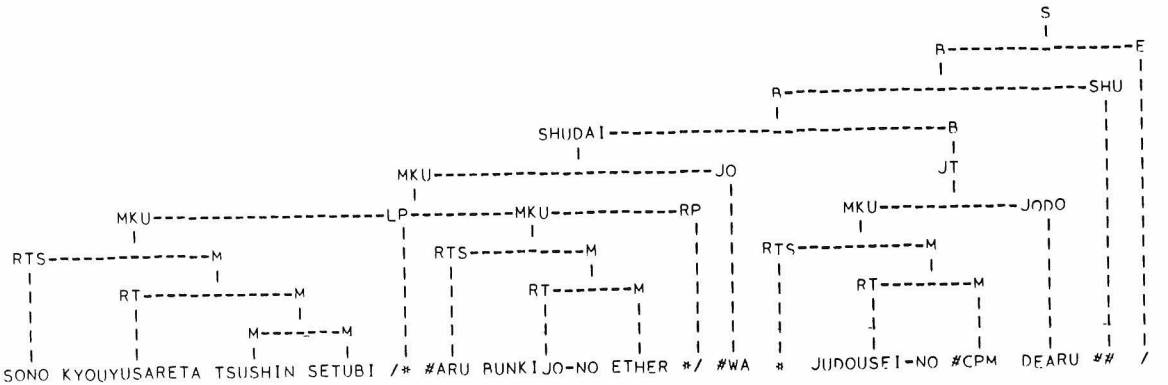
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (((** (A (BRANCHING ETHER)))
    (THE (SHARED ((#ADJ-CLSF COMMUNICATION) FACILITY))))
    (LAMBDA X66 ((A* (PASSIVE COMP)) (LAMBDA X67 (IS X66 X67))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



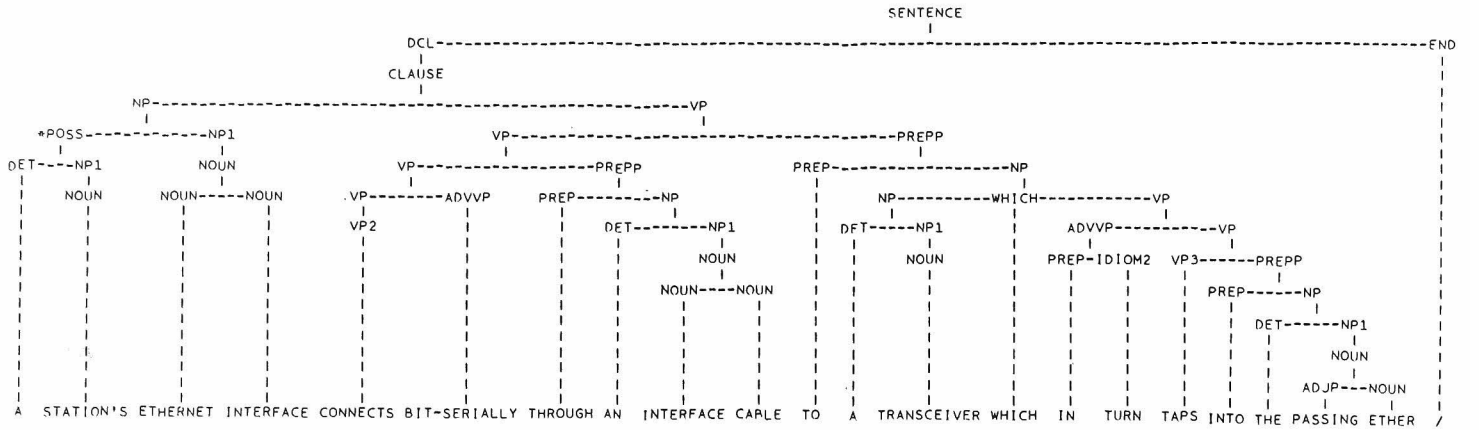
Input Sentence:

THE SHARED COMMUNICATION FACILITY * A BRANCHING ETHER * IS PASSIVE /

Output Sentence:

共有された通信設備（分岐状のETHER）は受動性のものである。

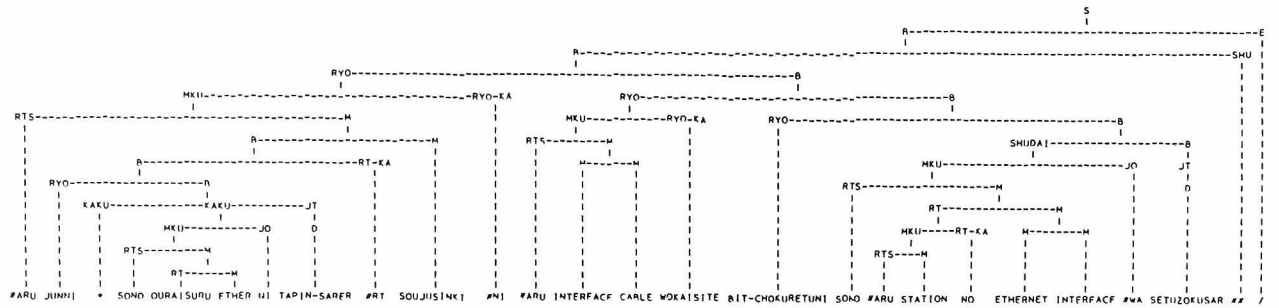
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((A STATION)
    (LAMBDA
      X37
      (((*POSS X37) ((*ADJ-CLSF ETHERNET) INTERFACE-N))
        (LAMBDA
          X48
          ((A
            ((WHICH
              (LAMBDA
                X45
                (IN-TURN
                  ((THE (PASSING ETHER)) (LAMBDA X46 (TAP-INTO X45 X46))))))
              TRANSCEIVER))
            (LAMBDA
              X47
              ((TO X47)
                ((A ((*ADJ-CLSF INTERFACE-N) CABLE))
                  (LAMBDA
                    X41
                    ((THROUGH X41) (BIT-SERIALLY (CONNECT-V2 X48))))))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



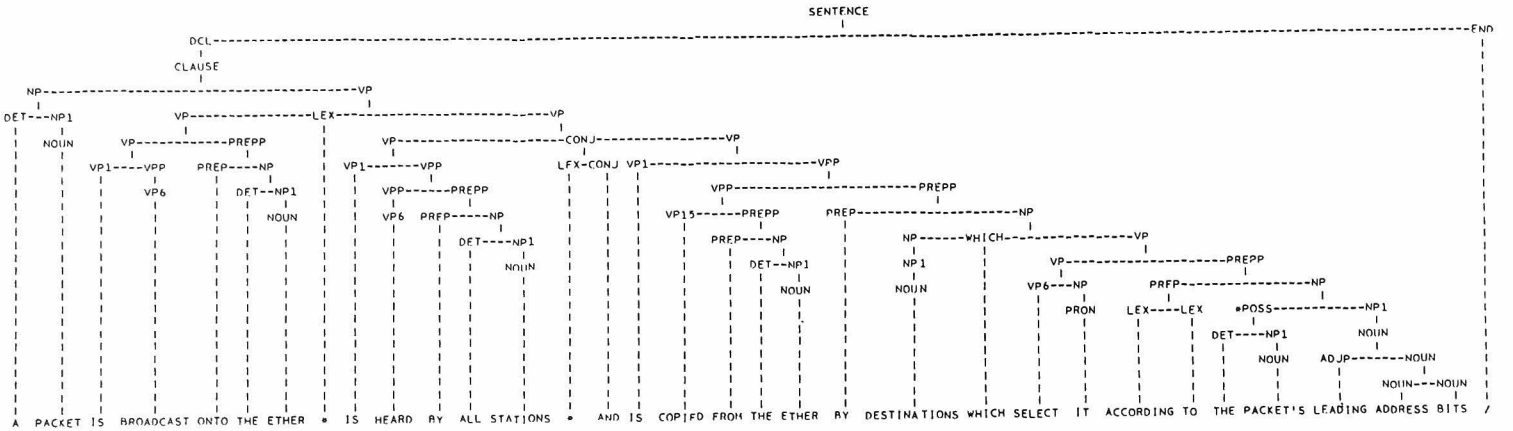
Input Sentence:

A STATION'S ETHERNET INTERFACE CONNECTS BIT-SERIALLY THROUGH AN INTERFACE CABLE TO A TRANSCEIVER WHICH IN TURN TAPS INTO THE PASSING ETHER /

Output Sentence:

ステーションのETHERNETインタフェースは順に往来するETHERにタップインされる送受信機にインタフェースケーブルを介してビット直列に接続される。

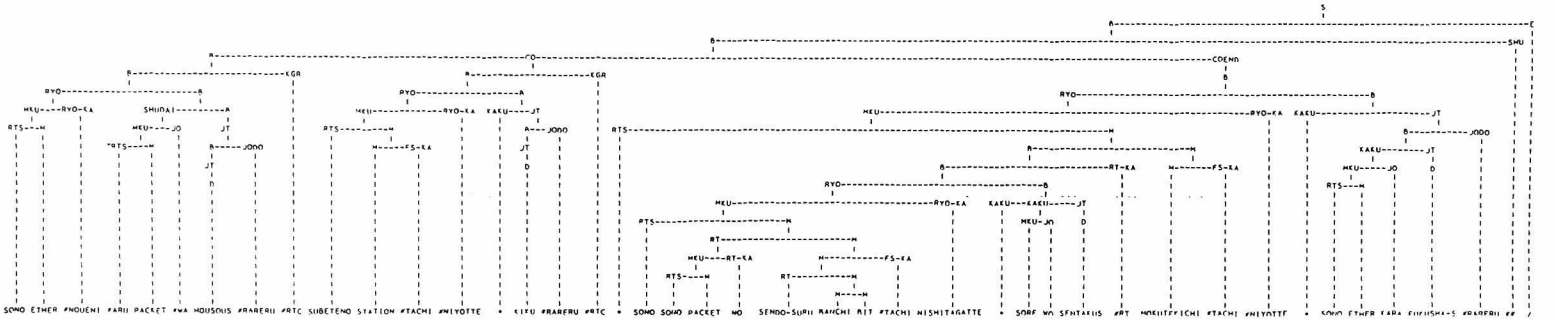
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((A PACKET)
    (AND*VP
      (LAMBDA
        X56
        ((THE ETHER)
          (LAMBDA X55 ((ONTO X55) ((*EN BROADCASTED) X56)))))
      (LAMBDA
        X59
        ((ALL (*PL STATION))
          (LAMBDA X58 ((*PSUBJ X58) ((*EN HEAR) X59)))))
      (LAMBDA
        X104
        ((A*
          ((WHICH
            (LAMBDA
              X83
              ((THE PACKET)
                (LAMBDA
                  X68
                  (((*POSS X68)
                    (*PL (LEADING ((*ADJ-CLSF ADDRESS) BIT-N))))
                    (LAMBDA
                      X82
                      ((ACCORDING-TO X82)
                        (IT (LAMBDA X64 (SELECT X83 X64))))))))))
            (*PL DESTINATION)))
          (LAMBDA
            X103
            ((*PSUBJ X103)
              ((THE ETHER) (LAMBDA X61 ((*EN COPY-FROM) X104 X61))))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



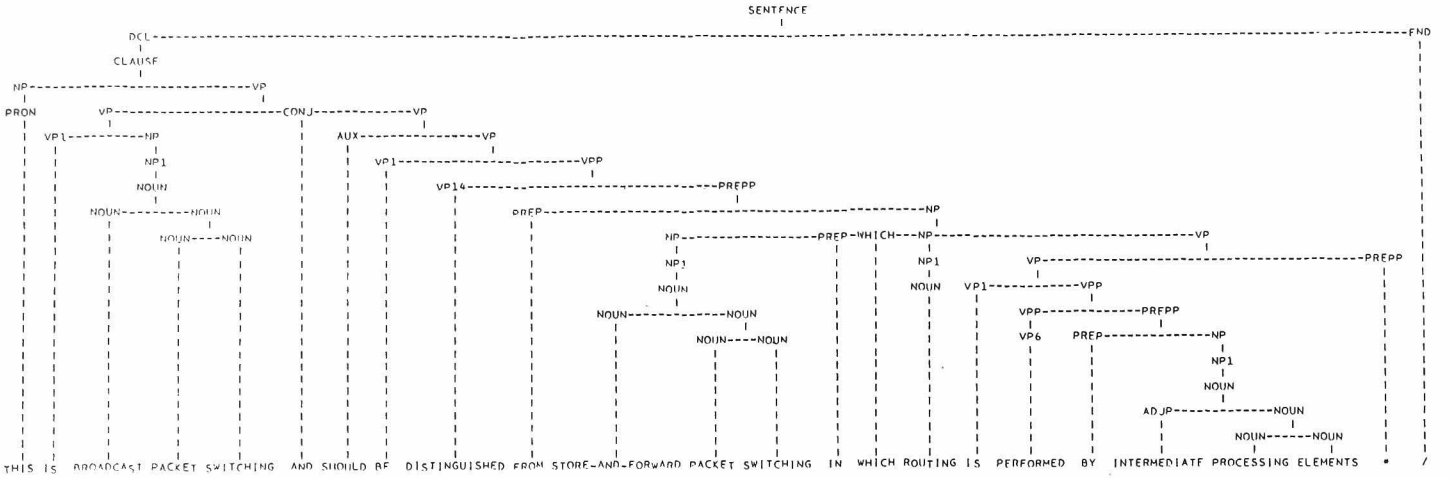
Input Sentence:

A PACKET IS BROADCAST ONTO THE ETHER * IS HEARD BY ALL STATIONS * AND IS COPIED FROM THE ETHER BY DESTINATIONS WHICH SELECT IT ACCORDING TO THE PACKET'S LEADING ADDRESS BITS /

Output Sentence:

パケットはそのＥＴＨＥＲの上に放送され，全てのステーションによって聞かれ，そのパケットの先導する番地ｂｉｔに従ってそれを選択する目的地によってそのＥＴＨＥＲから複写される。

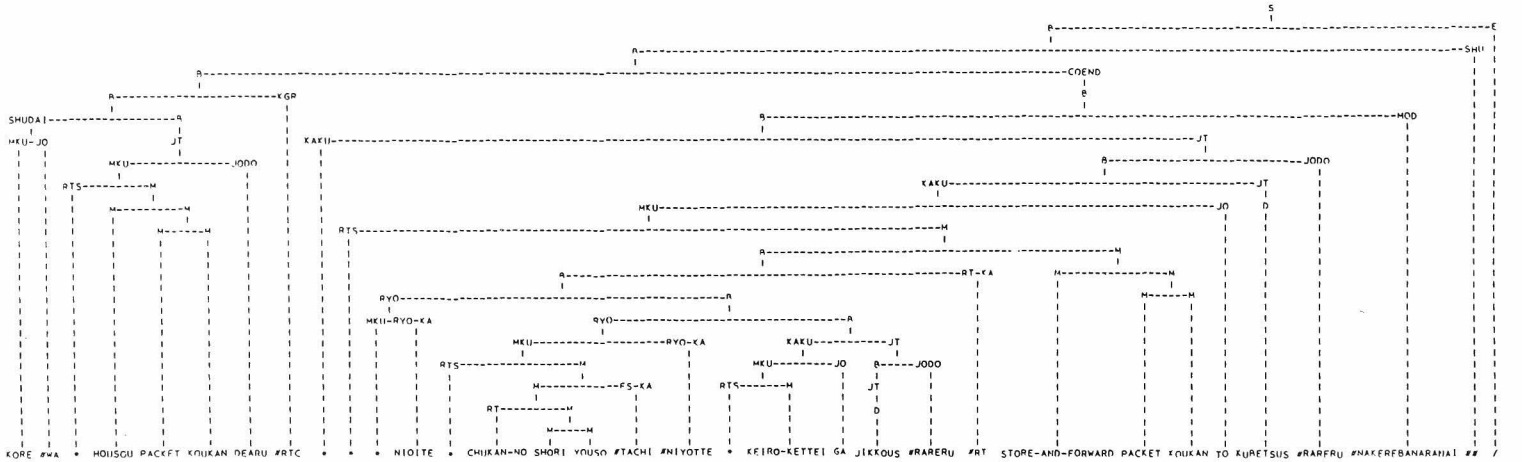
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (THIS
    (AND*VP
      (LAMBDA
        X140
        ((*AB ((*ADJ-CLSF BROADCAST) ((*ADJ-CLSF PACKET) SWITCHING)))
          (LAMBDA X141 (IS X140 X141))))
      (SHOULD
        (LAMBDA
          X74
          ((*AB
            ((WHICH
              (LAMBDA
                X49
                ((*AB ROUTING)
                  (LAMBDA
                    X73
                    ((IN X49)
                      ((A*
                        (*PL
                          (INTERMEDIATE ((*ADJ-CLSF PROCESSING) ELEMENT))))
                        (LAMBDA X70 ((*PSUBJ X70) ((*EN PERFORM) X73)))))))))
              ((*ADJ-CLSF STORE-AND-FORWARD)
                ((*ADJ-CLSF PACKET) SWITCHING))))
            (LAMBDA X75 ((*EN3 DISTINGUISH-FROM) X74 X75)))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



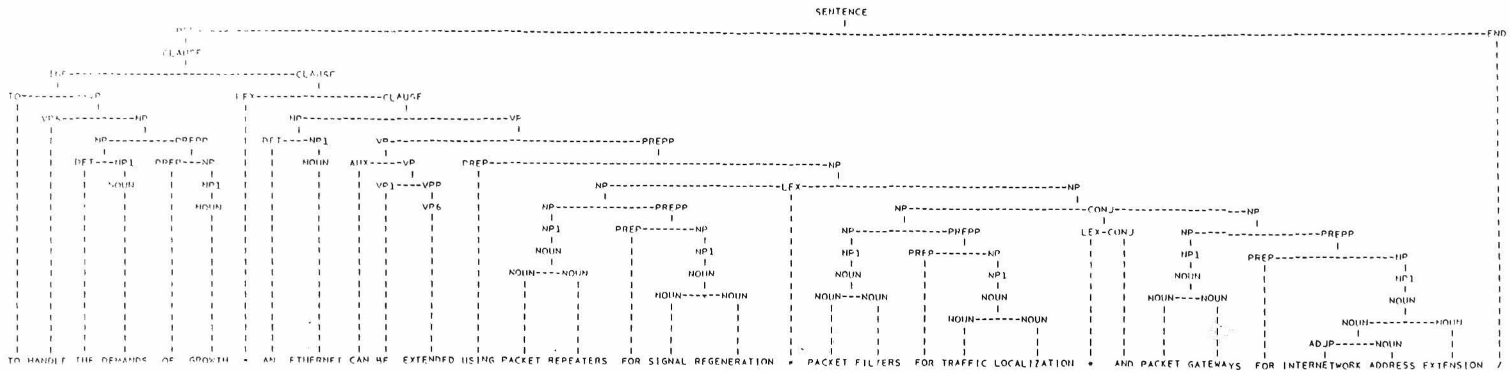
Input Sentence:

THIS IS BROADCAST PACKET SWITCHING AND SHOULD BE DISTINGUISHED FROM STORE-AND-FORWARD PACKET SWITCHING IN WHICH ROUTING IS PERFORMED BY INTERMEDIATE PROCESSING ELEMENTS /

Output Sentence:

これは放送パケット交換であり，中間の処理要素によって経路決定が実行されるストア・アンド・フォワードパケット交換と区別されなければならない。

Result of Phrase Structure Analysis:



Extracted EFR Expression:

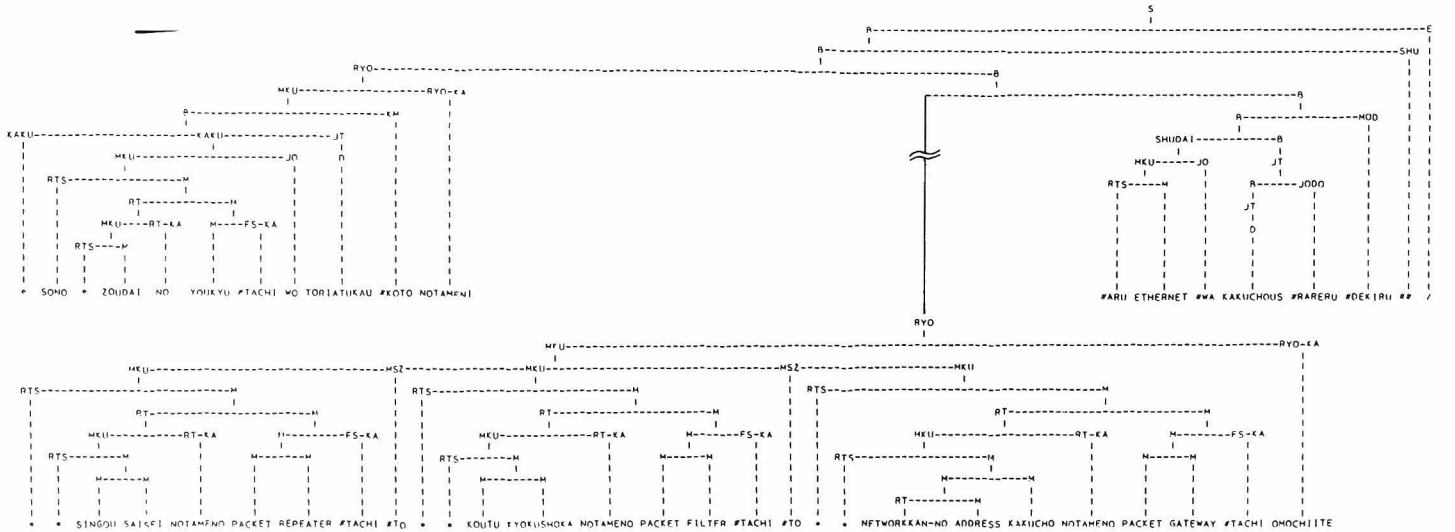
```
(#DCL
  ((INF
    (LAMBDA
      X148
      ((THE
        (LAMBDA
          X147
          ((*AB GROWTH)
            (LAMBDA X146 ((((*AP OF) X146) (*PL DEMAND)) X147)))))
        (LAMBDA X149 (HANDLE X148 X149)))))
   (LAMBDA
     X151
     ((INF-CL X151)
       ((A ETHERNET)
         (LAMBDA
           X137
           ((*AND
             (A*
               (LAMBDA
                 X131
                 ((*AB ((*ADJ-OBJ SIGNAL) REGENERATION))
                   (LAMBDA
                     X130
                     ((((*AP FOR) X130) (*PL ((*ADJ-OBJ PACKET) REPEATER)))

                       X131)))))
            (A*
              (LAMBDA
                X91
                ((*AB ((*ADJ-OBJ TRAFFIC) LOCALIZATION))
                  (LAMBDA
                    X90
                    ((((*AP FOR) X90) (*PL ((*ADJ-CLSF PACKET) FILTER)))

                      X91)))))
            (A*
              (LAMBDA
                X88
                ((*AB ((*ADJ-OBJ (INTERNETWORK ADDRESS)) EXTENSION))
                  (LAMBDA
                    X87
                    ((((*AP FOR) X87) (*PL ((*ADJ-CLSF PACKET) GATEWAY)))

                      X88))))))
           (LAMBDA
             X136
             ((USING X136)
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



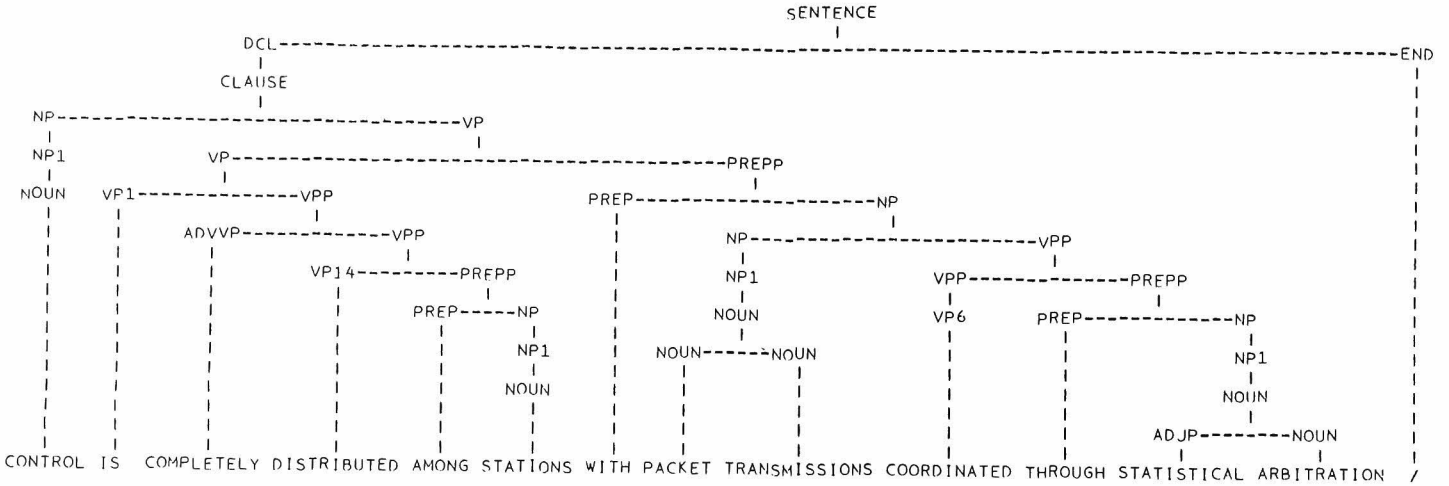
Input Sentence:

TO HANDLE THE DEMANDS OF GROWTH * AN ETHERNET CAN BE EXTENDED USING PACKET REPEATERS FOR SIGNAL REGENERATION * PACKET FILTERS FOR TRAFFIC LOCALIZATION * AND PACKET GATEWAYS FOR INTERNETWORK ADDRESS EXTENSION /

Output Sentence:

増大の要求を取り扱うために信号再生のためのパケットリピータと交通局所化のためのパケットフィルターとネットワーク間のアドレス拡張のためのパケットゲートウェイを用いてETHERNETは拡張されることができる。

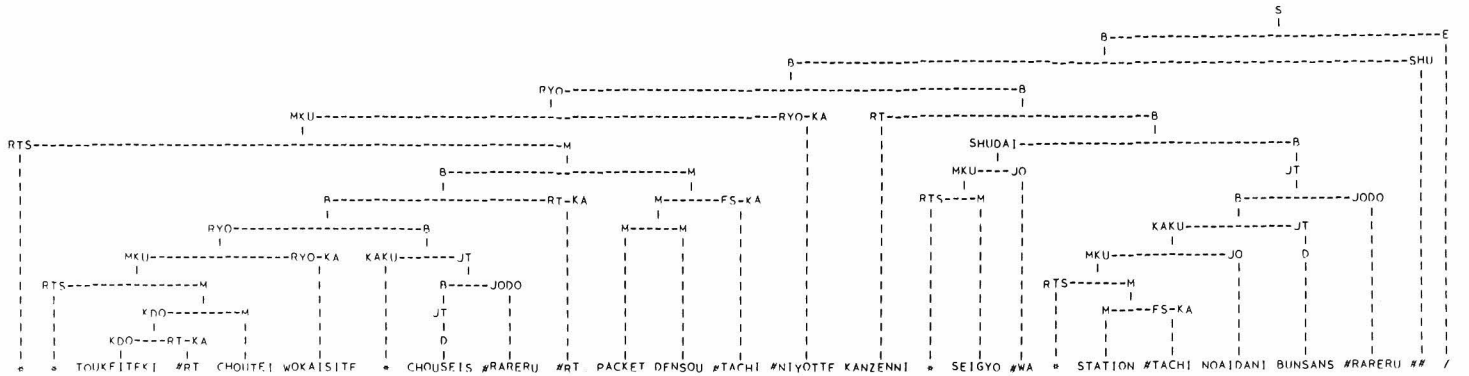
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((*AB CONTROL-N)
    (LAMBDA
      X21
      ((A*
        ((PARTI
          (LAMBDA
            X19
            ((*AB (STATISTICAL ARBITRATION))
              (LAMBDA
                X18
                ((THROUGH X18) ((*EN COORDINATE) X19))))))
          (*PL ((*ADJ-OBJ PACKET) TRANSMISSION))))
        (LAMBDA
          X20
          ((WITH X20)
            (COMPLETELY
              ((A* (*PL STATION))
                (LAMBDA
                  X10
                  ((*EN3 DISTRIBUTE-AMONG) X21 X10)))))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



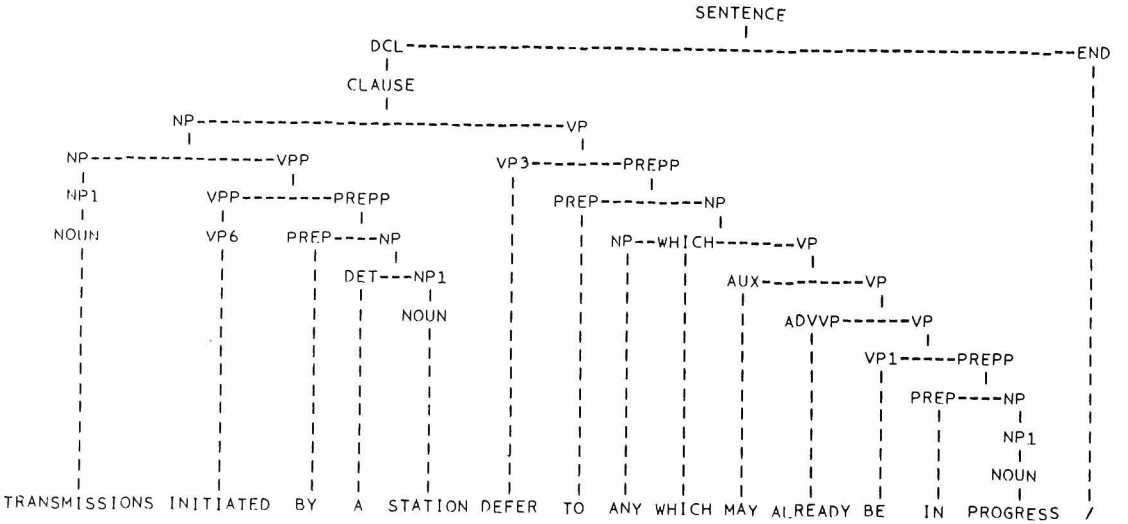
Input Sentence:

CONTROL IS COMPLETELY DISTRIBUTED AMONG STATIONS WITH PACKET TRANSMISSIONS COORDINATED THROUGH STATISTICAL ARBITRATION /

Output Sentence:

制御は統計的調停を介して調整されるパケット伝送で完全にステーションの間に分散される。

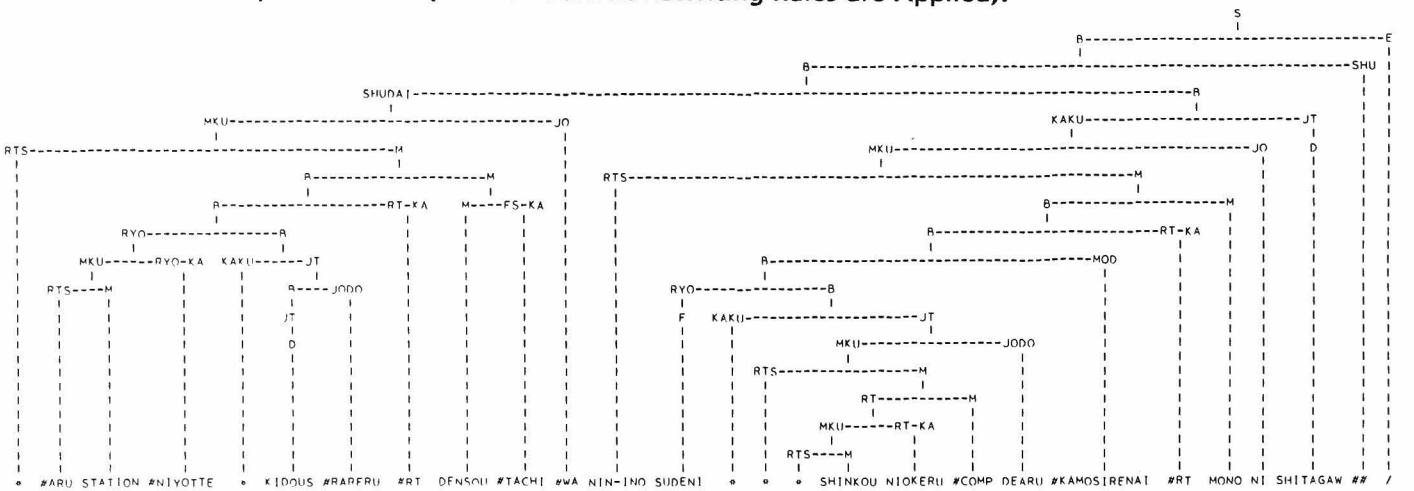
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((A*
    ((PARTI
      (LAMBDA
        X9
        ((A STATION) (LAMBDA X8 ((*PSUBJ X8) ((*EN INITIATE) X9)))))))
     (*PL TRANSMISSION)))
    (LAMBDA
      X16
      ((ANY*
        ((WHICH
          (MAY
            (LAMBDA
              X14
              (ALREADY
                ((A*
                  (LAMBDA
                    X13
                    ((*AB PROGRESS)
                      (LAMBDA X12 ((((*AP IN) X12) COMP) X13)))))
                  (LAMBDA X15 (IS X14 X15)))))))
          THING))
        (LAMBDA X17 (DEFER-TO X16 X17)))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



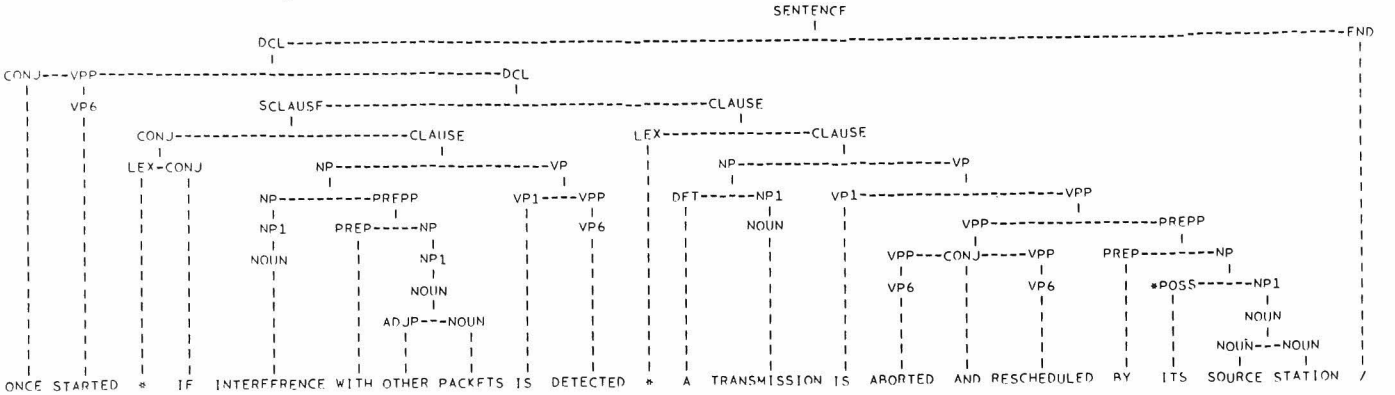
Input Sentence:

TRANSMISSIONS INITIATED BY A STATION DEFER TO ANY WHICH MAY ALREADY BE IN PROGRESS /

Output Sentence:

ステーションによって起動される伝送は任意の既に進行におけるものであるかもしれないものに従う。

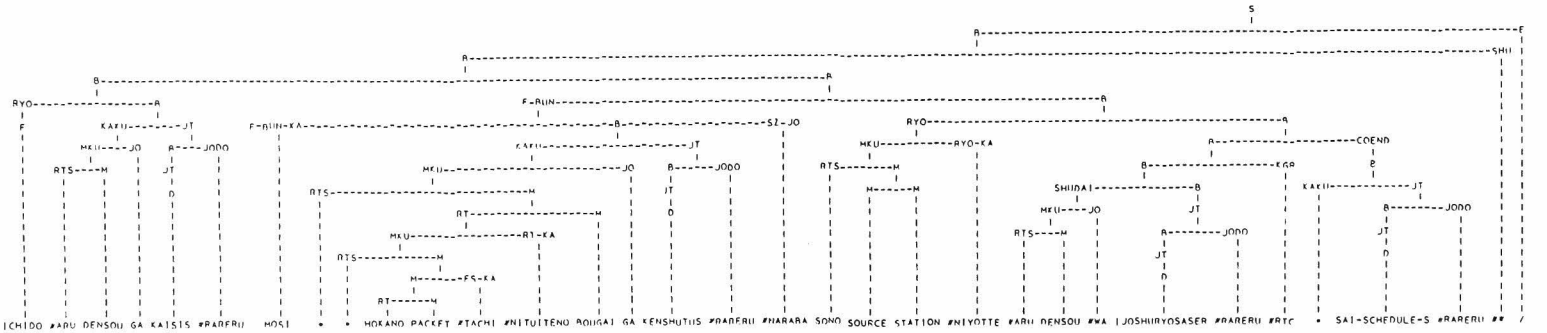
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((A TRANSMISSION)
    (LAMBDA
      X16
      ((ONCE-C (#PRES ((*EN START-VT) X16)))
        ((IF
          ((*AB
            (LAMBDA
              X4
              ((A* (*PL (OTHER PACKET)))
                (LAMBDA X3 ((((*AP WITH4) X3) INTERFERENCE) X4)))))
            (LAMBDA X5 ((*EN DETECT) X5))))
          ((LAMBDA
            X14
            (((ITS ((*ADJ-CLSF SOURCE) STATION))
              (LAMBDA
                X13
                ((*PSUBJ X13)
                  ((AND*VP
                    (LAMBDA X6 ((*EN ABORT) X6))
                    (LAMBDA X9 ((*EN RESCHEDULE) X9)))
                    X14)))))
            X16))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



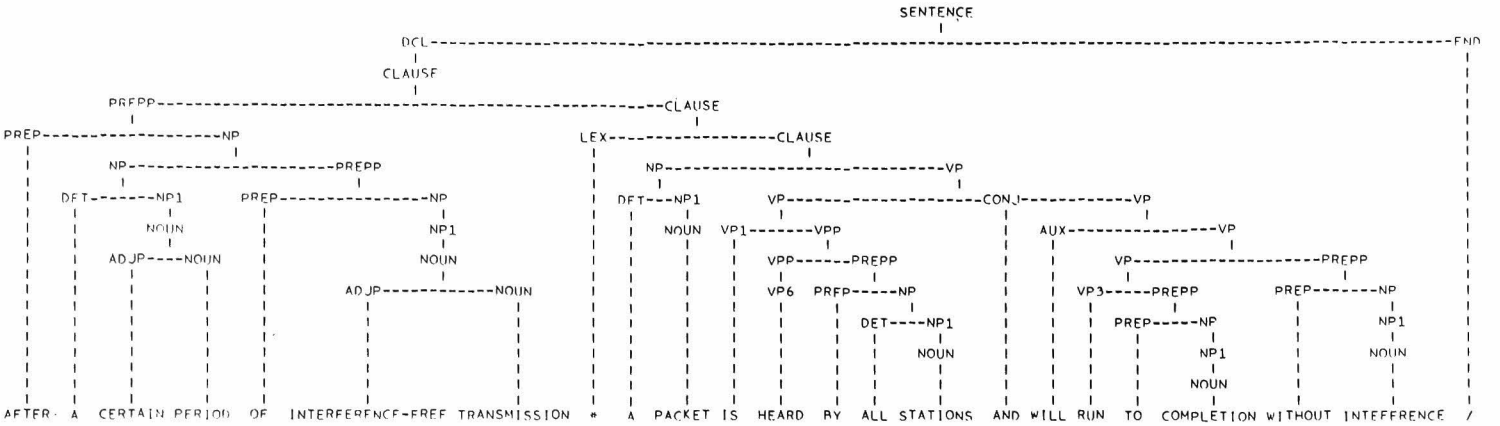
Input Sentence:

ONCE STARTED * IF INTERFERENCE WITH OTHER PACKETS IS DETECTED * A TRANSMISSION IS ABORTED AND RESCHEDULED BY ITS SOURCE STATION /

Output Sentence:

一度伝送が開始されもし他のパケットについての妨害が検出されるならばそのソースステーションによって伝送は異常終了させられ，再スケジュールされる。

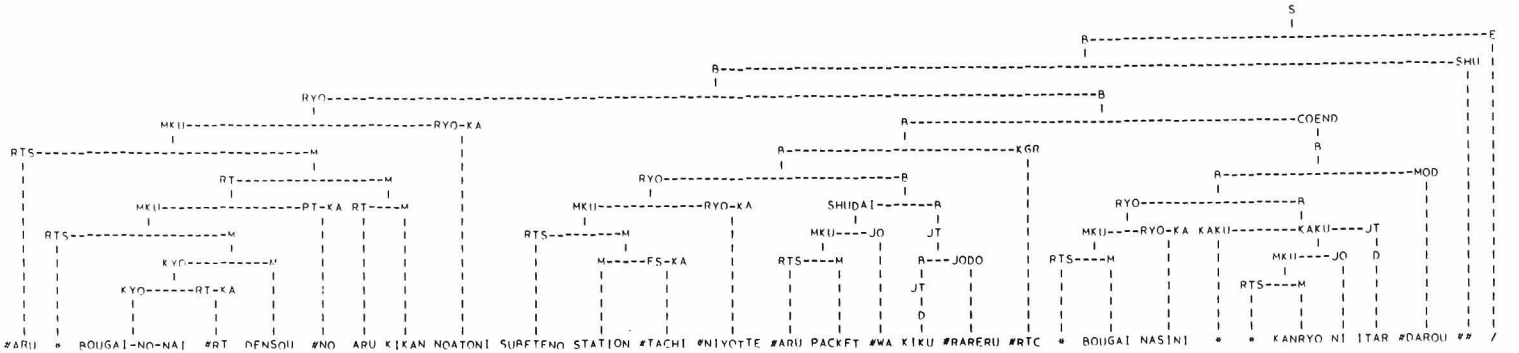
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((A
     (LAMBDA
       X35
       ((*AB (INTERFERENCE-FREE TRANSMISSION))
         (LAMBDA X34 ((((*AP OF) X34) (CERTAIN PERIOD)) X35)))))
     (LAMBDA
       X38
       ((AFTER X38)
         ((A PACKET)
           (AND*VP
             (LAMBDA
               X6
               ((ALL (*PL STATION))
                 (LAMBDA X5 ((*PSUBJ X5) ((*EN HEAR) X6)))))
             (WILL
               (LAMBDA
                 X19
                 ((*AB INTERFERENCE)
                   (LAMBDA
                     X18
                     ((WITHOUT X18)
                       ((*AB COMPLETION) (LAMBDA X10 (RUN-TO X19 X10)))))))))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



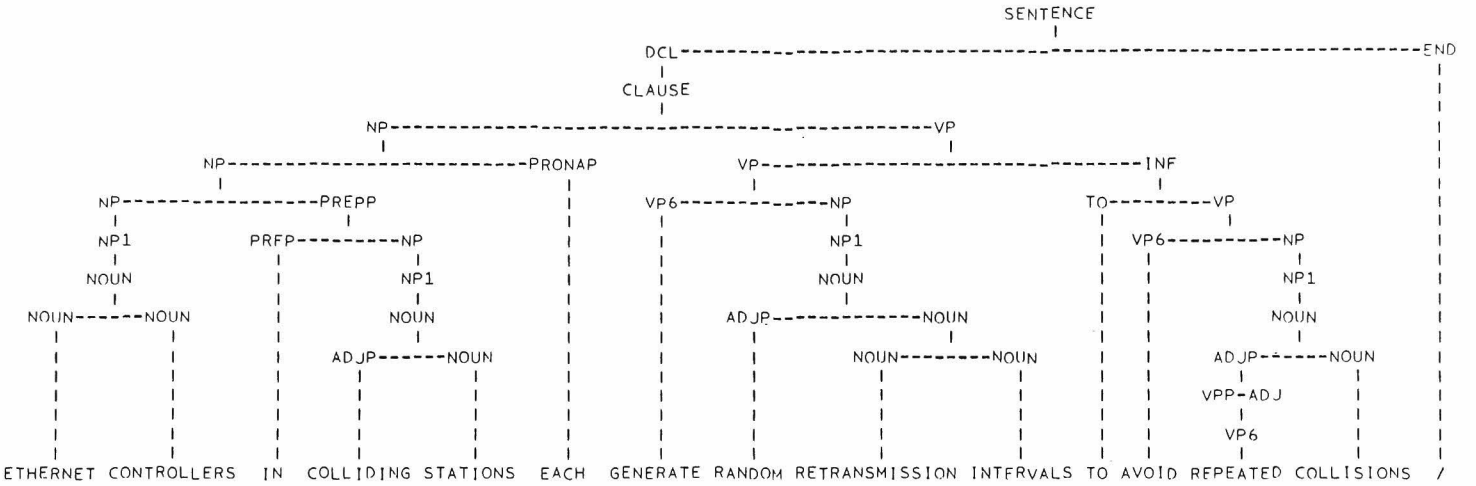
Input Sentence:

AFTER A CERTAIN PERIOD OF INTERFERENCE-FREE TRANSMISSION * A PACKET IS HEARD BY ALL STATIONS AND WILL RUN TO COMPLETION WITHOUT INTEFERENCE /

Output Sentence:

妨害のない伝送の或る期間の後にパケットは全てのステーションによって聞かれ，妨害なしに完了に至るだろう。

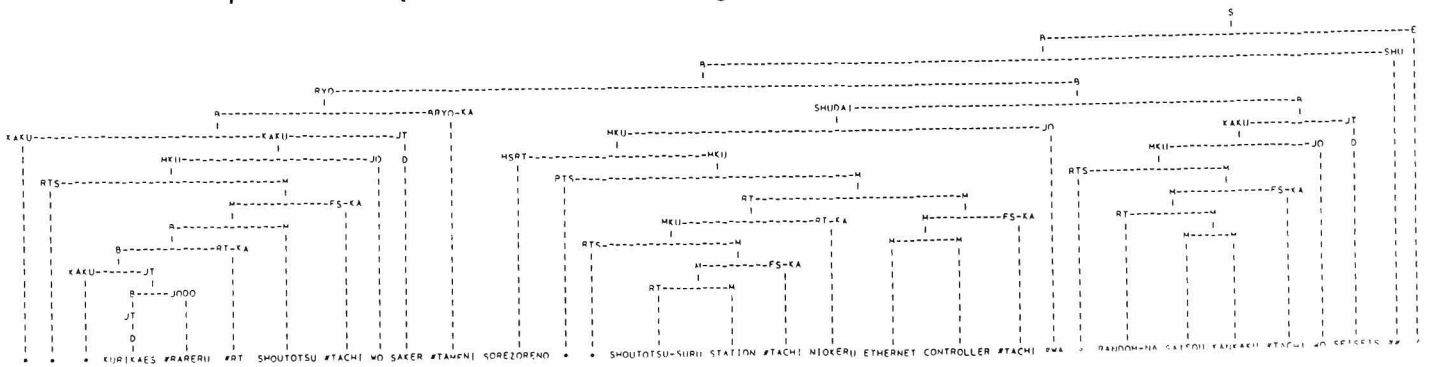
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((EACH-PRONAP
    (A*
      (LAMBDA
        X5
        ((A* (*PL (COLLIDING STATION)))
          (LAMBDA
            X4
            ((((*AP IN) X4) (*PL ((*ADJ-CLSF ETHERNET) CONTROLLER)))
              X5))))))
    (LAMBDA
      X67
      (((INF-AV
        (LAMBDA
          X18
          ((A* (*PL ((PART (*EN REPEAT)) COLLISION)))
            (LAMBDA X19 (AVOID X18 X19)))))
        (LAMBDA
          X65
          ((A* (*PL (RANDOM ((*ADJ RETRANSMISSION) INTERVAL))))
            (LAMBDA X66 (GENERATE X65 X66)))))
        X67))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



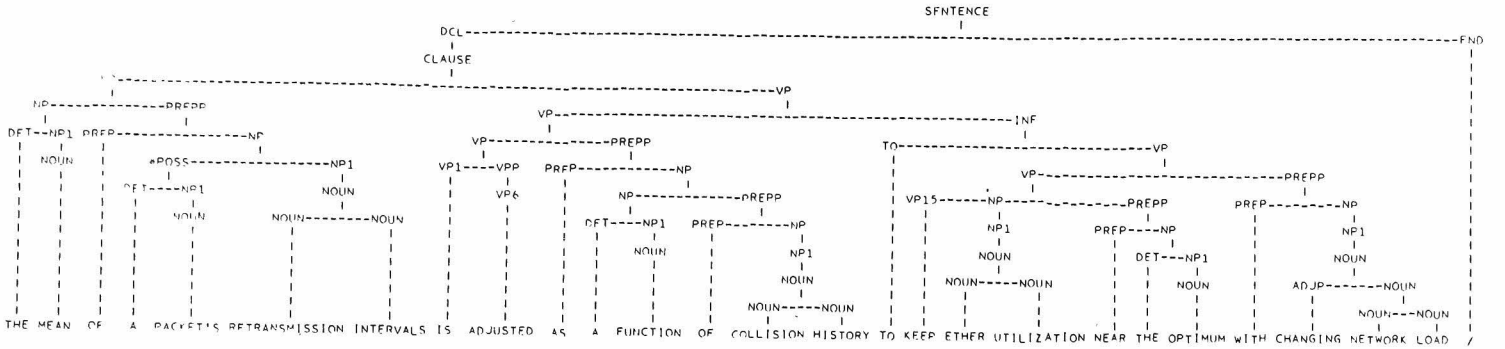
Input Sentence:

ETHERNET CONTROLLERS IN COLLIDING STATIONS EACH GENERATE RANDOM RETRANSMISSION INTERVALS TO AVOID REPEATED COLLISIONS /

Output Sentence:

それぞれの衝突するステーションにおけるETHERNETコントローラは繰り返される衝突を避けるためにランダムな再送間隔を生成する。

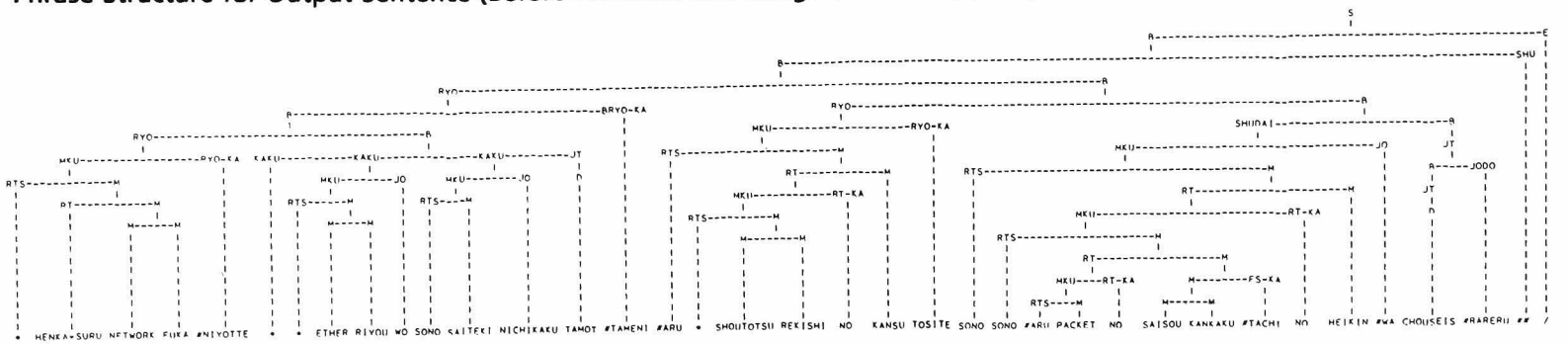
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((THE
    (LAMBDA
      X9
      ((A PACKET)
        (LAMBDA
          X5
          ((((*POSS X5) (*PL ((*ADJ RETRANSMISSION) INTERVAL)))
            (LAMBDA X8 ((((*AP OF) X8) MEAN) X9)))))))))
   (LAMBDA
     X94
     (((INF-AV
       (LAMBDA
         X18
         ((*AB (CHANGING ((*ADJ-CLSF NETWORK) LOAD)))
           (LAMBDA
             X17
             ((WITH X17)
               ((*AB ((*ADJ-ORJ ETHER) UTILIZATION))
                 (LAMBDA
                   X13
                   ((THE OPTIMUM) (LAMBDA X14 (KEEP-NEAR X18 X13 X14)))))))))))
      (LAMBDA
        X89
        ((A
          (LAMBDA
            X11
            ((*AB ((*ADJ-CLSF COLLISION) HISTORY))
              (LAMBDA X10 ((((*AP OF) X10) FUNCTION-N3) X11)))))))
       (LAMBDA X88 ((AS X88) ((*EN ADJUST) X89)))))))
   X94))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



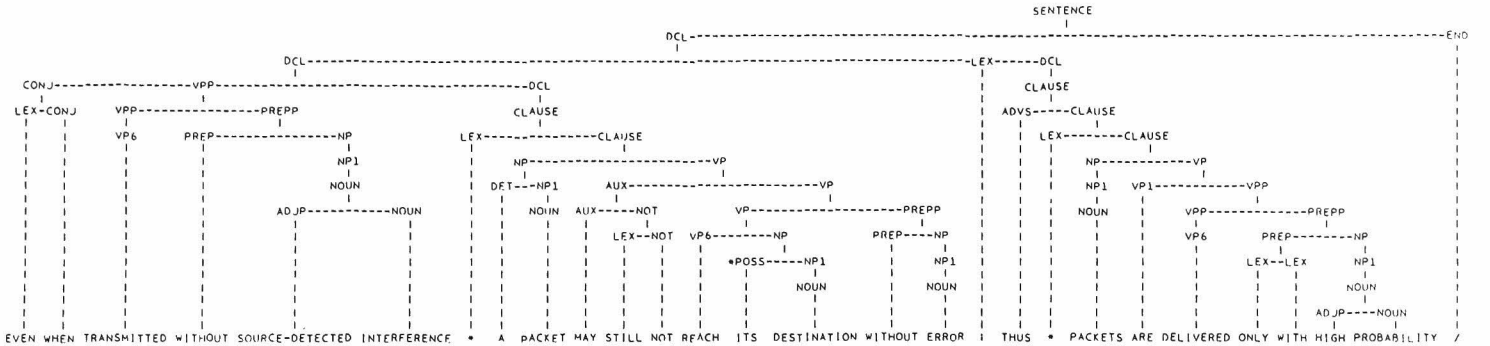
Input Sentence:

THE MEAN OF A PACKET'S RETRANSMISSION INTERVALS IS ADJUSTED AS A FUNCTION OF COLLISION HISTORY TO KEEP ETHER UTILIZATION NEAR THE OPTIMUM WITH CHANGING NETWORK LOAD /

Output Sentence:

パケットの再送間隔の平均は変化するネットワーク負荷でETHER利用をその最適に保つために衝突歴史の関数として調整される。

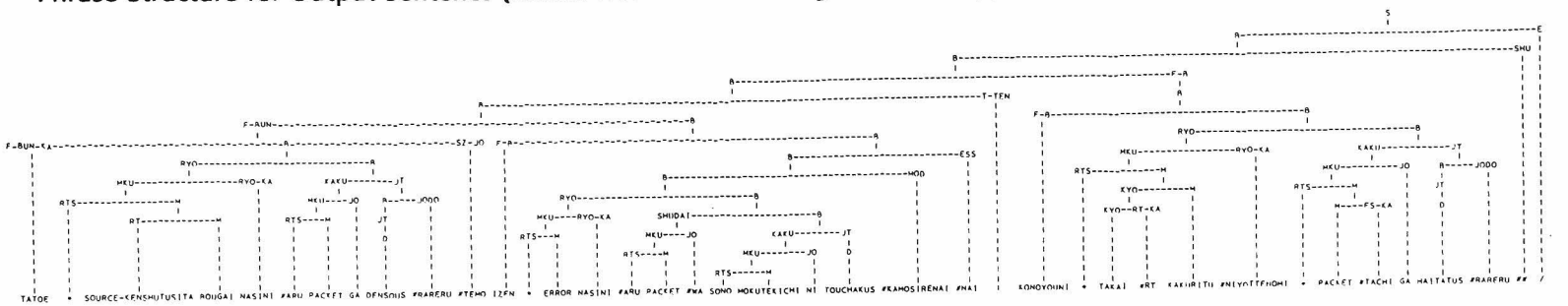
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
 (I
   ((A PACKET)
     (LAMBDA
       X36
       ((EVEN-IF
         (#PRES
           ((*AB (SOURCE-DETECTED INTERFERENCE))
             (LAMBDA X34 ((WITHOUT X34) ((*EN TRANSMIT) X36))))))
         ((STILL-NOT
           (MAY
             (LAMBDA
               X31
               ((*AB ERROR)
                 (LAMBDA
                   X30
                   ((WITHOUT X30)
                     ((ITS DESTINATION) (LAMBDA X18 (REACH X31 X18)))))))))
           X36))))
   (THUS
     ((A* (*PL PACKET))
       (LAMBDA
         X29
         ((*AB (HIGH PROBABILITY))
           (LAMBDA X28 ((ONLY-WITH X28) ((*EN DELIVER) X29))))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



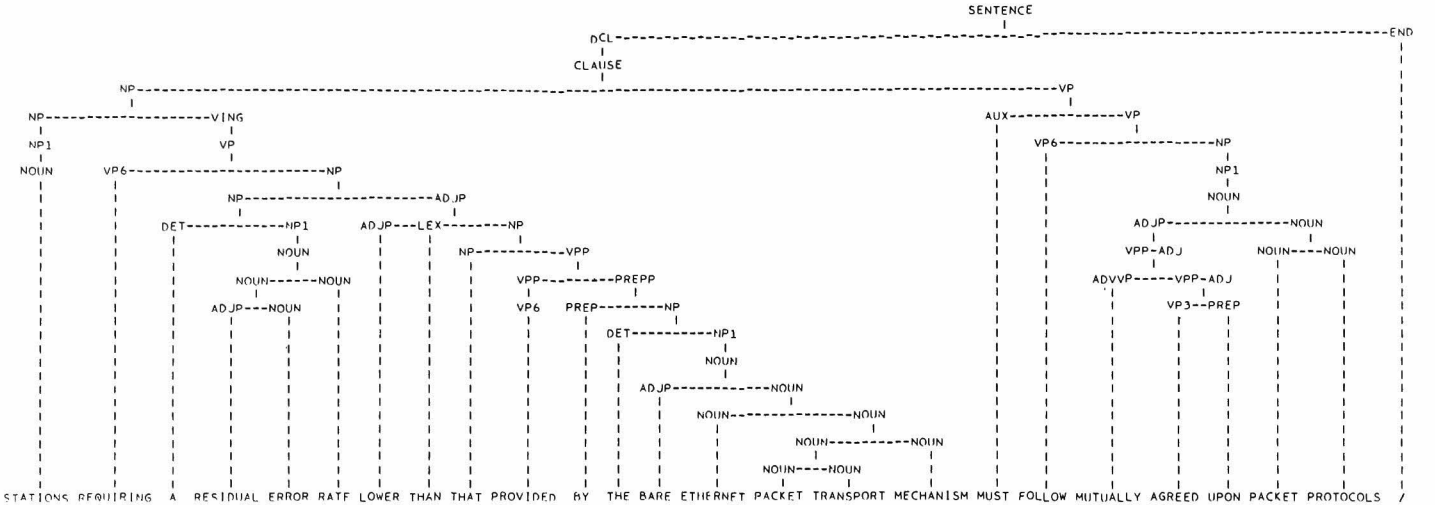
Input Sentence:

EVEN WHEN TRANSMITTED WITHOUT SOURCE-DETECTED INTERFERENCE * A PACKET MAY STILL NOT REACH ITS DESTINATION WITHOUT ERROR ; THUS * PACKETS ARE DELIVERED ONLY WITH HIGH PROBABILITY /

Output Sentence:

たとえソース側で検出された妨害なしにパケットが伝送されても依然パケットはエラーなしにその目的地に到着しないかもしれない；このように高い確率によってのみパケットが配達される。

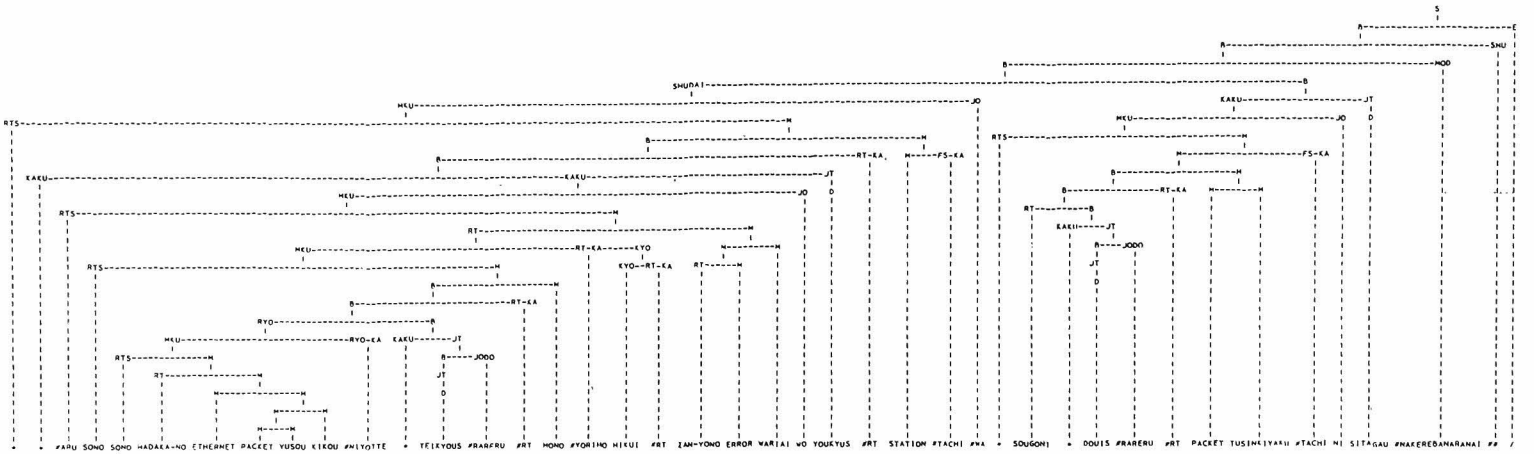
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((A*
    ((PART!
      (LAMBDA
        X66
        ((A
          (LAMBDA
            X22
            ((THAT**
              ((PART!
                (LAMBDA
                  X11
                  ((THE
                    (BARE
                      ((*ADJ-CLSF ETHERNET)
                        ((*ADJ-CLSF ((*ADJ-CLSF PACKET) TRANSPORT))
                          MECHANISM))))
                      (LAMBDA X10 ((*PSUBJ X10) ((*EN PROVIDE) X11))))))
                THING))
              (LAMBDA
                X23
                ((((MORE-X-THAN LOW) X23)
                  ((*ADJ-CLSF (RESIDUAL ERROR)) RATE))
                  X22)))))
          (LAMBDA X67 (REQUIRE-1 X66 X67)))))
      (*PL STATION)))
    (MUST
      (LAMBDA
        X110
        ((A*
          (*PL
            ((PART! (MUTUALLY (*EN AGREE-ON)))
              ((*ADJ-CLSF PACKET) PROTOCOL))))
          (LAMBDA X111 (FOLLOW1 X110 X111)))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



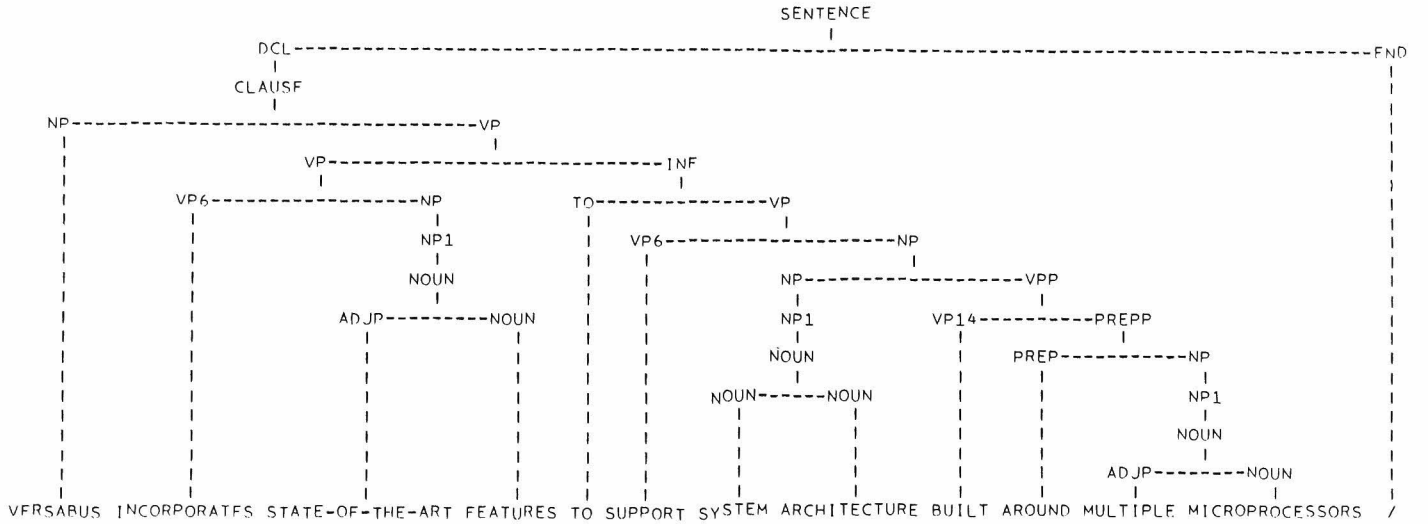
Input Sentence:

STATIONS REQUIRING A RESIDUAL ERROR RATE LOWER THAN THAT PROVIDED BY THE BARE ETHERNET PACKET TRANSPORT MECHANISM MUST FOLLOW MUTUALLY AGREED UPON PACKET PROTOCOLS /

Output Sentence:

裸のETHERNETパケット輸送機構によって提供されるものよりも低い残余のエラー割合を要求するステーションは相互に同意されるパケット通信規約に従わなければならない。

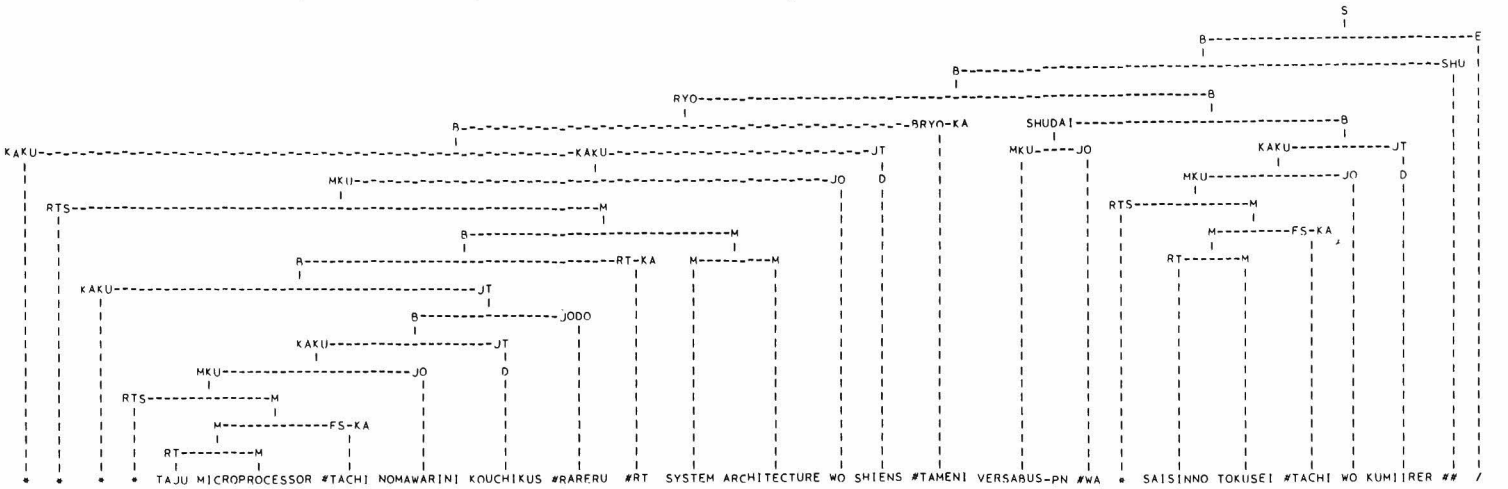
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (VERSABUS-PN
    (LAMBDA
      X332
      (((INF-AV
        (LAMBDA
          X330
          ((*AB
            ((PARTI
              (LAMBDA
                X328
                ((A* (*PL (MULTIPLE MICROPROCESSOR)))
                  (LAMBDA X329 ((*EN3 BUILD-AROUND) X328 X329))))))
              ((*ADJ-CLSF SYSTEM) ARCHITECTURE)))
            (LAMBDA X331 (SUPPORT-V1 X330 X331)))))
        (LAMBDA
          X322
          ((A* (*PL (STATE-OF-THE-ART FEATURE)))
            (LAMBDA X323 (INCORPORATE X322 X323)))))
        X332))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



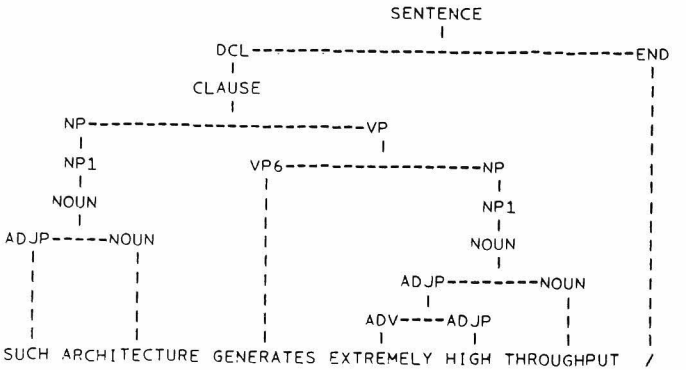
Input Sentence:

VERSABUS INCORPORATES STATE-OF-THE-ART FEATURES TO SUPPORT SYSTEM ARCHITECTURE BUILT AROUND MULTIPLE MICROPROCESSORS /

Output Sentence:

VERSAbusは多重マイクロプロセッサの周りに構築されるシステムア－キテクチャを支援するために最新の特性を組込む。

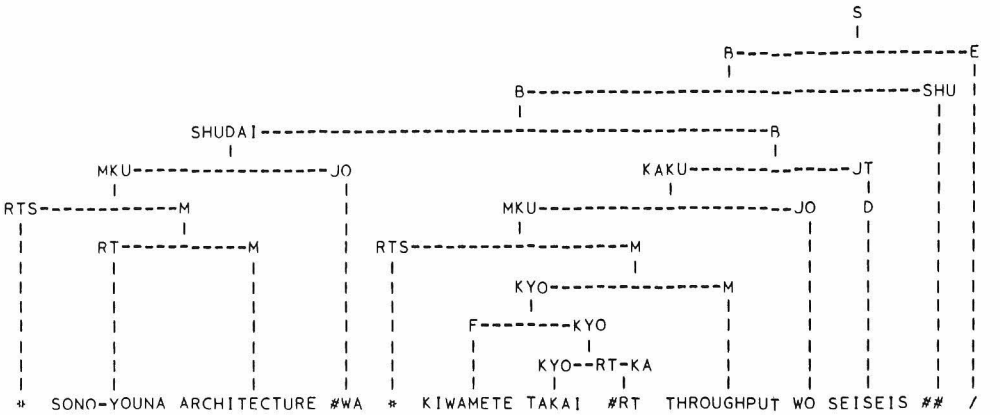
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((*AB (SUCH ARCHITECTURE))
    (LAMBDA
      X70
      ((*AB ((EXTREMELY HIGH) THROUGHPUT))
        (LAMBDA X71 (GENERATE X70 X71))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



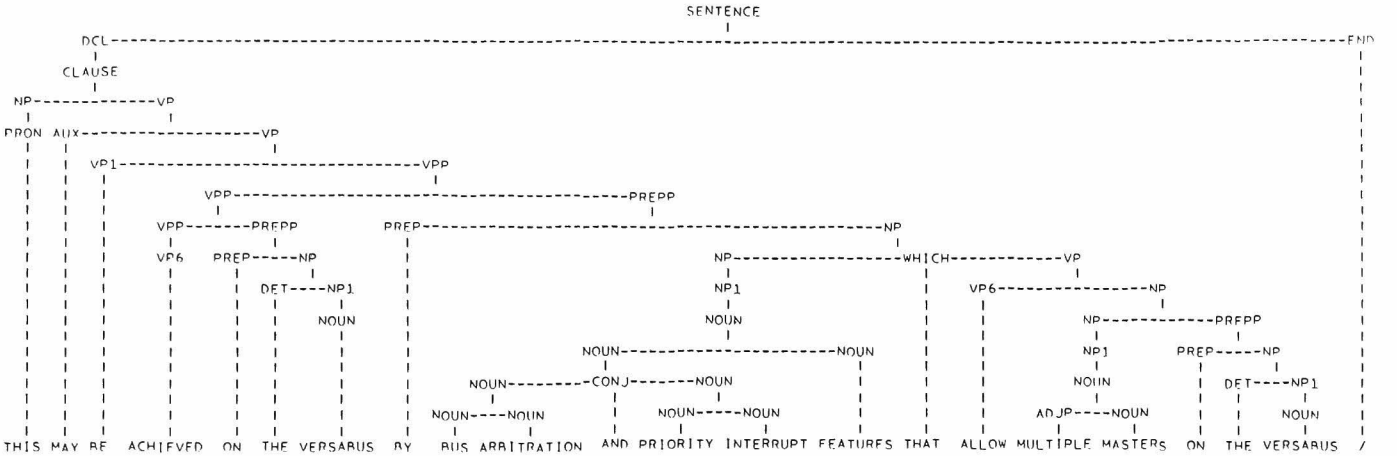
Input Sentence:

SUCH ARCHITECTURE GENERATES EXTREMELY HIGH THROUGHPUT /

Output Sentence:

そのようなアーキテクチャは極めて高いスループットを生成する。

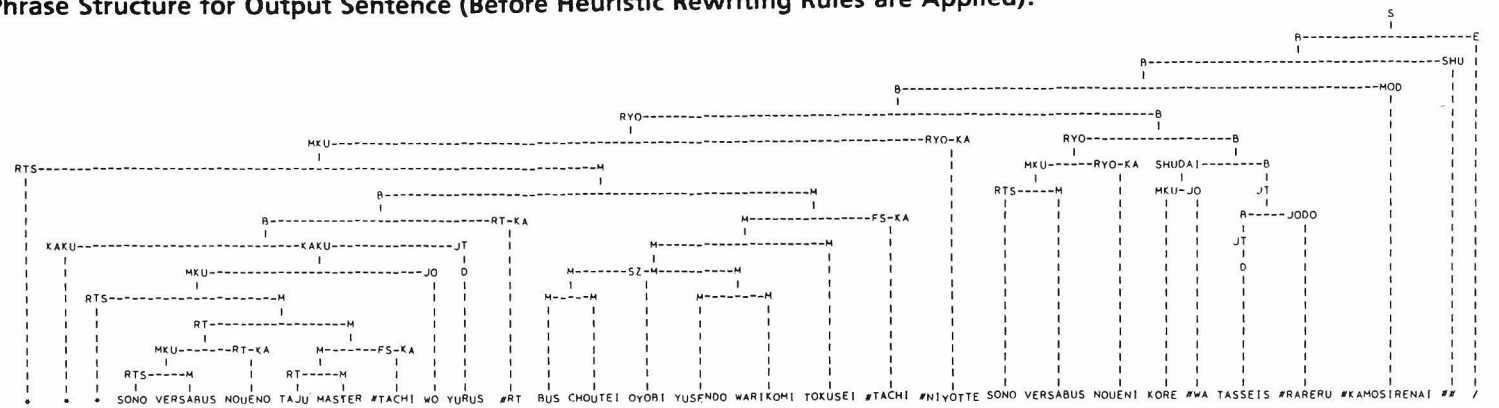
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (THIS
    (MAY
      (LAMBDA
        X161
        ((A*
          ((WHICH
            (LAMBDA
              X30
              ((A*
                (LAMBDA
                  X29
                  ((THE VERSABUS)
                    (LAMBDA
                      X28
                      ((((*AP ON) X28)
                        (*PL (MULTIPLE MASTER)))
                       X29))))))
                (LAMBDA X31 (ALLOW X30 X31)))))
           (*PL
             ((*ADJ-AP
               (*AND-NOUN-NOUN
                 ((*ADJ-OBJ BUS) ARBITRATION)
                 ((*ADJ-CLSF PRIORITY) INTERRUPT-N)))
              FEATURE))))
         (LAMBDA
           X160
           ((*PSUBJ X160)
             ((THE VERSABUS)
               (LAMBDA
                 X108
                 ((ON X108) ((*EN ACHIEVE) X161)))))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



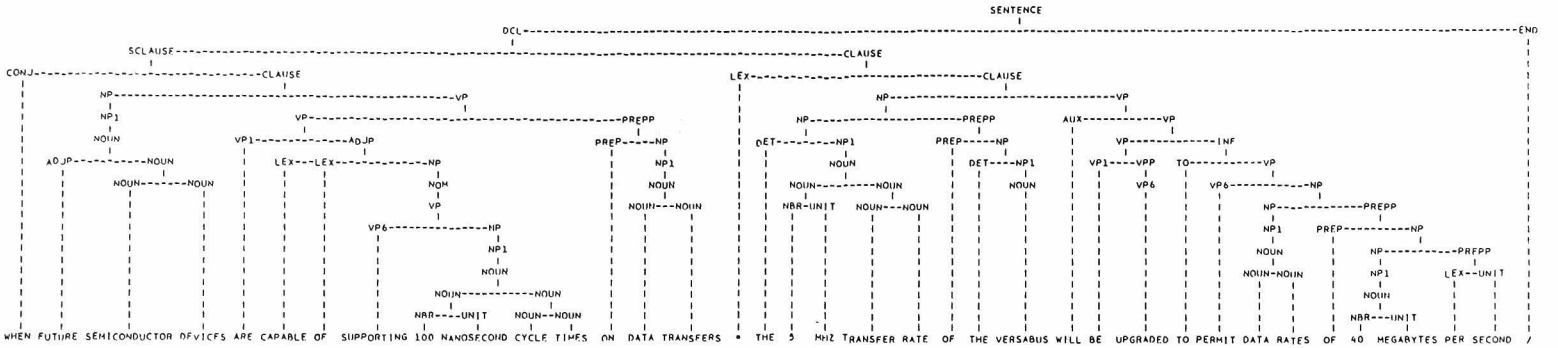
Input Sentence:

THIS MAY BE ACHIEVED ON THE VERSABUS BY BUS ARBITRATION AND PRIORITY INTERRUPT FEATURES THAT ALLOW MULTIPLE MASTERS ON THE VERSABUS /

Output Sentence:

これはそのVERSAbusの上の多重マスターを許すバス調停及び優先度割込み特性によってそのVERSAbusの上に達成されるかもしれない。

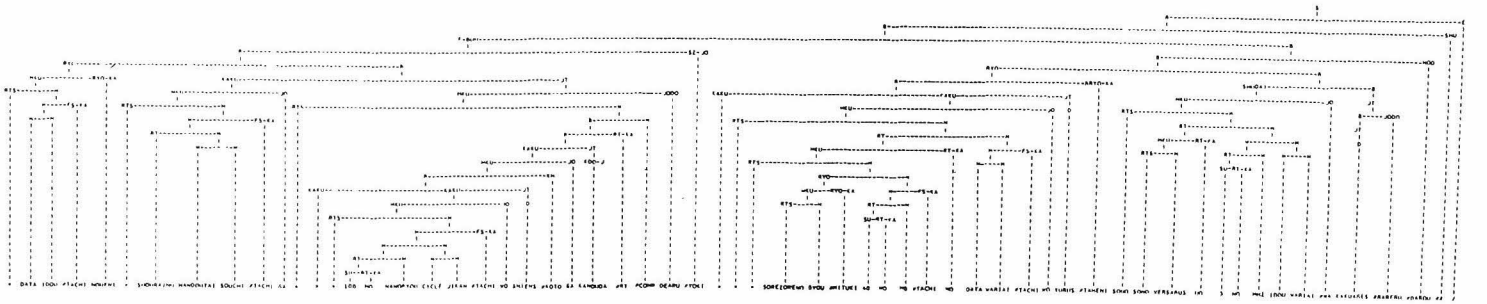
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  ((WHEN
     ((A* (*PL (FUTURE-A ((*ADJ-CLSF SEMICONDUCTOR) DEVICE))))
       (LAMBDA
         X573
         ((A* (*PL ((*ADJ-OBJ DATA) TRANSFER)))
           (LAMBDA
             X572
             ((ON X572)
               ((A*
                 (LAMBDA
                   X568
                   ((NOM
                     (LAMBDA
                       X565
                       ((A*
                         (*PL
                           ((*ADJ-CLSF ((*NBR (QUOTE 100)) NANOSECOND))
                             ((*ADJ-CLSF CYCLE) TIME))))
                         (LAMBDA X566 (SUPPORT-V1 X565 X566)))))
                    (LAMBDA X567 (((CAPABLE-OF X567) COMP) X568)))))
                (LAMBDA X571 (IS X573 X571)))))))))
   ((THE
      (LAMBDA
        X577
        ((THE VERSABUS)
          (LAMBDA
            X576
            ((((*AP OF) X576)
              ((*ADJ-CLSF ((*NBR (QUOTE 5)) MHZ))
                ((*ADJ-CLSF TRANSFER) RATE)))
              X577)))))
     (WILL
       (LAMBDA
         X630
         (((INF-AV
            (LAMBDA
              X601
              ((A*
                (LAMBDA
                  X600
                  ((A*
                    (LAMBDA
                      X587
                      ((EACH SECOND)
                        (LAMBDA
                          X586
                          ((((*AP PER) X586)
                            (*PL ((*NBR (QUOTE 40)) MEGARYTE)))
                            X587)))))
                    (LAMBDA
                      X599
                      ((((*AP OF) X599) (*PL ((*ADJ-CLSF DATA) RATE)))
                        X600)))))
               (LAMBDA X602 (PERMIT X601 X602)))))
           (LAMBDA X580 ((*EN UPGRADE) X580)))
          X630)))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



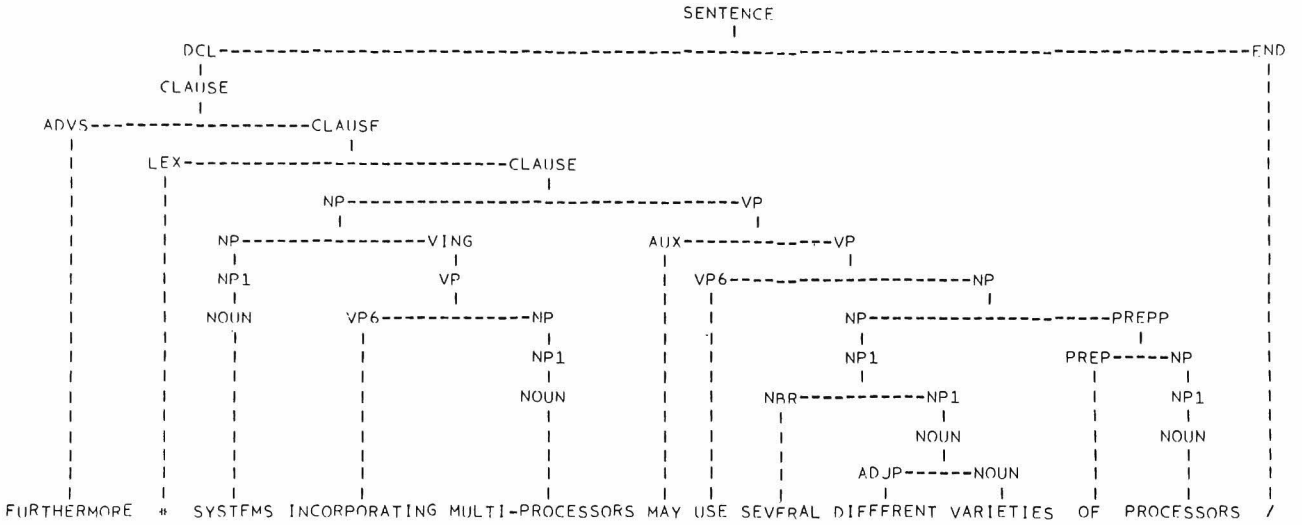
Input Sentence:

WHEN FUTURE SEMICONDUCTOR DEVICES ARE CAPABLE OF SUPPORTING 100 NANOSECOND CYCLE TIMES ON DATA TRANSFERS * THE 5 MHZ TRANSFER RATE OF THE VERSABUS WILL BE UPGRADED TO PERMIT DATA RATES OF 40 MEGABYTES PER SECOND /

Output Sentence:

データ移動の上に将来の半導体装置が１００ナノ秒サイクル時間を支援することが可能なものであるときそのＶＥＲＳＡｂｕｓの５ＭＨｚ移動割合はそれぞれの秒につき４０ＭＢのデータ割合を許すために格上げされるだろう。

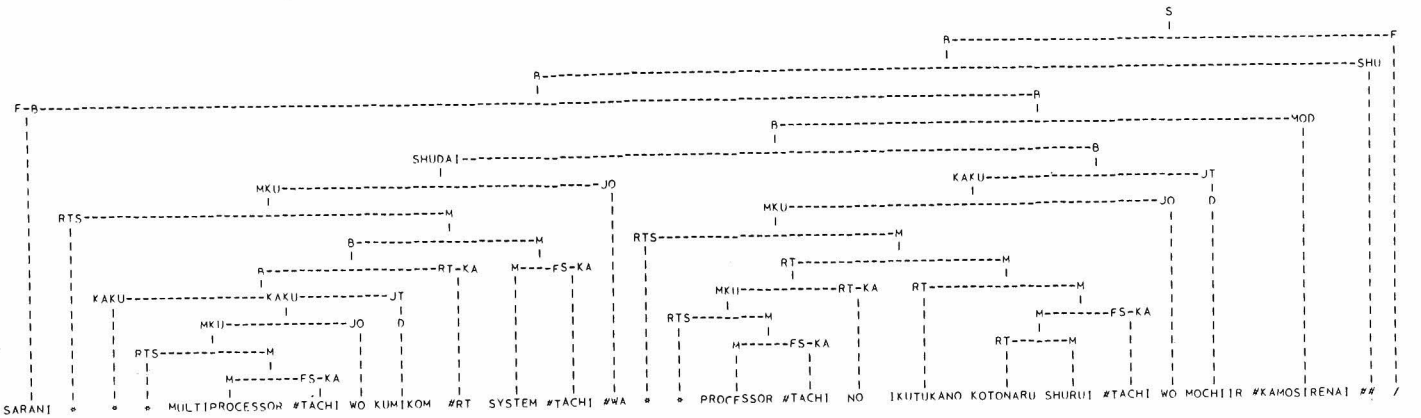
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (FURTHERMORE
    ((A*
      ((PARTI
        (LAMBDA
          X6
          ((A* (#PL MULTI-PROCESSOR))
            (LAMBDA X7 (INCORPORATE X6 X7)))))
        (#PL SYSTEM)))
      (MAY
        (LAMBDA
          X12
          ((A*
            (LAMBDA
              X11
              ((A* (#PL PROCESSOR))
                (LAMBDA
                  X10
                  ((((#AP OF) X10) (SEVERAL (#PL (DIFFERENT VARIETY))))
                    X11)))))
            (LAMBDA X13 (USE-V1 X12 X13)))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



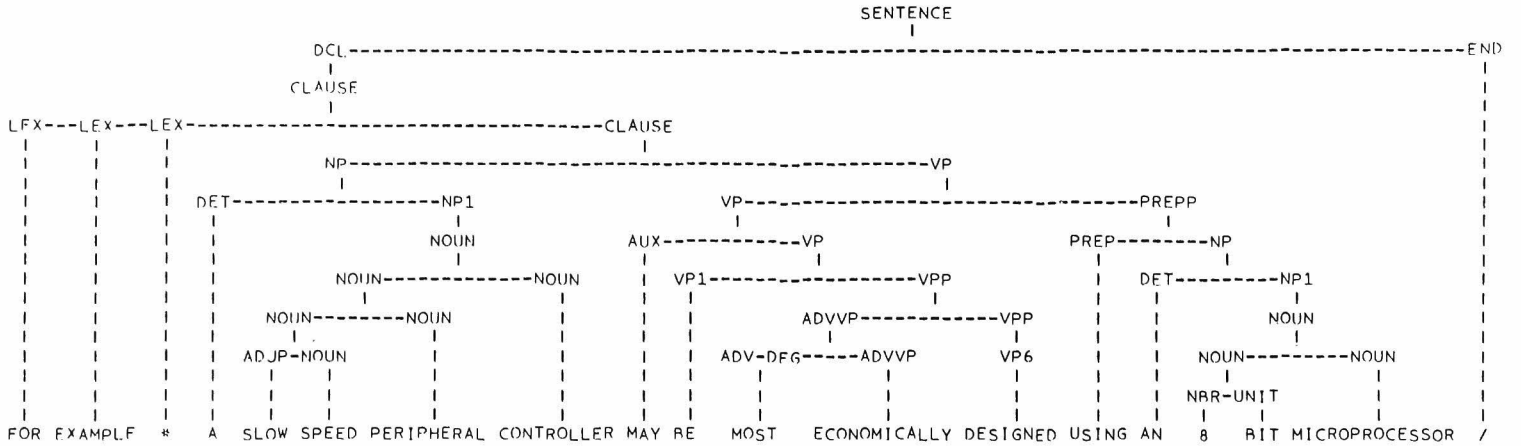
Input Sentence:

FURTHERMORE * SYSTEMS INCORPORATING MULTI-PROCESSORS MAY USE SEVERAL DIFFERENT VARIETIES OF PROCESSORS /

Output Sentence:

更にマルチプロセッサを組込むシステムはいくつかの異なる種類のプロセッサを用いるかもしれない。

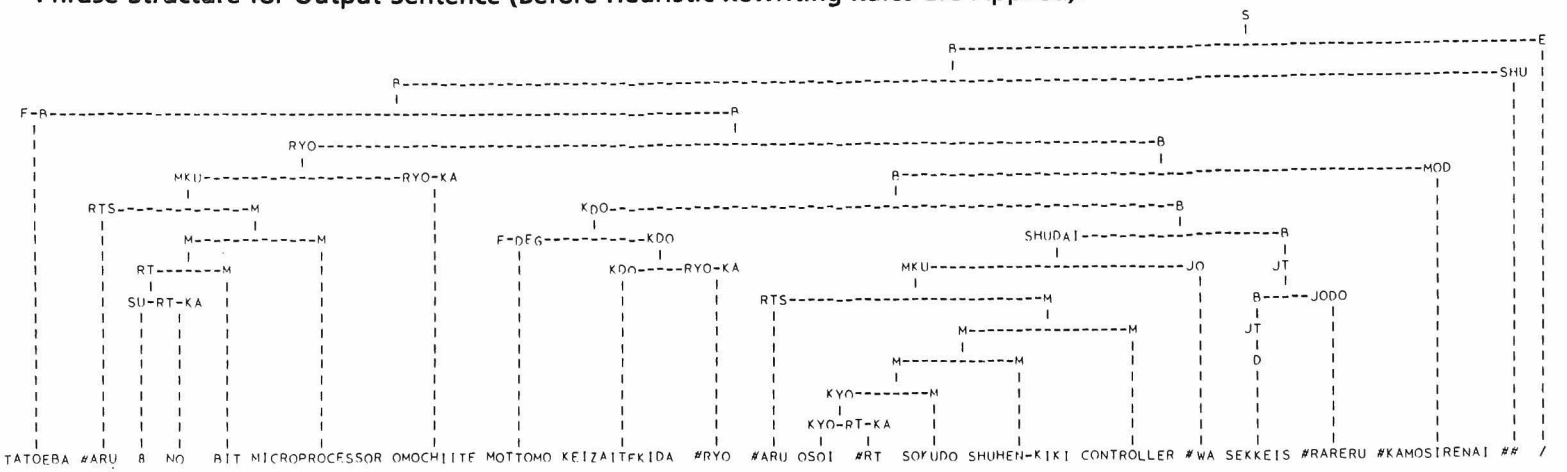
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (EG*S
    ((A
      ((*ADJ-CLSF ((*ADJ-CLSF (SLOW SPEED)) PERIPHERAL))
        CONTROLLER))
      (LAMBDA
        X6
        ((A ((*ADJ-CLSF ((*NBR (QUOTE 8)) BIT)) MICROPROCESSOR))
          (LAMBDA
            X5
            ((USING X5)
              ((MAY
                 (LAMBDA X4 ((MOST ECONOMICALLY) ((*EN DESIGN-V) X4))))
                X6))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



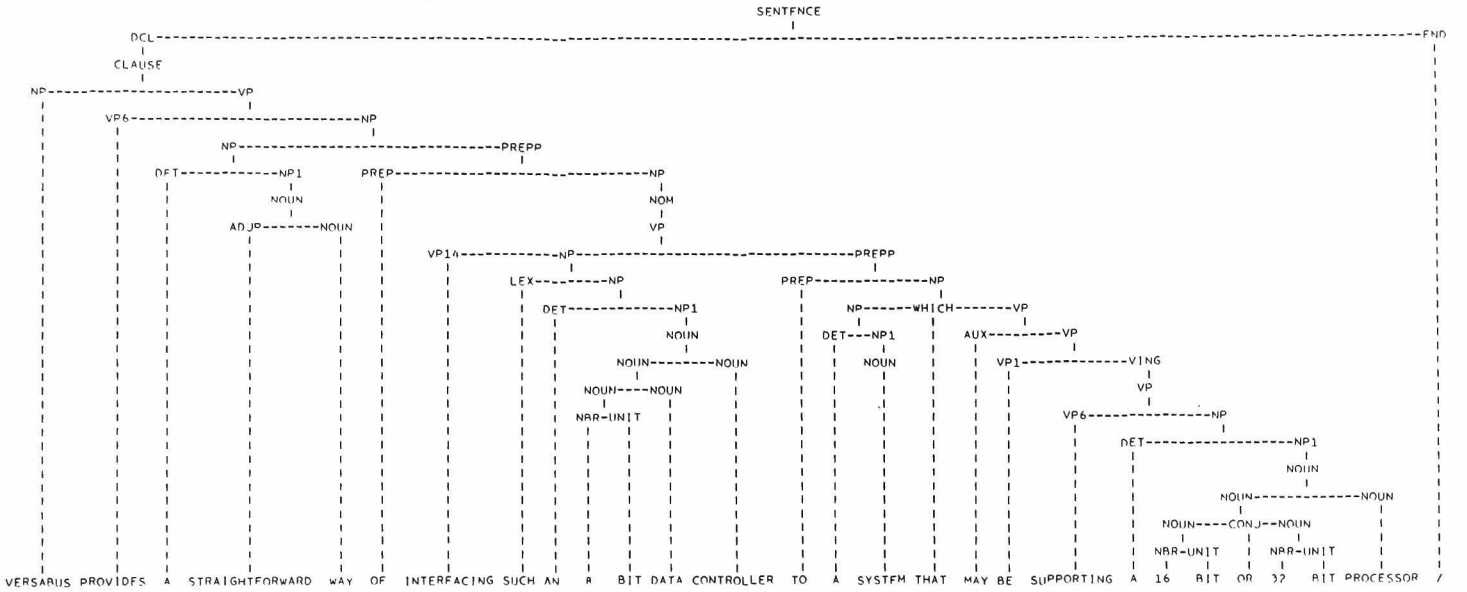
Input Sentence:

FOR EXAMPLE * A SLOW SPEED PERIPHERAL CONTROLLER MAY BE MOST ECONOMICALLY DESIGNED USING AN 8 BIT MICROPROCESSOR /

Output Sentence:

例えば８ｂｉｔマイクロプロセッサを用いて最も経済的に遅い速度の周辺機器コントローラは設計されるかもしれない。

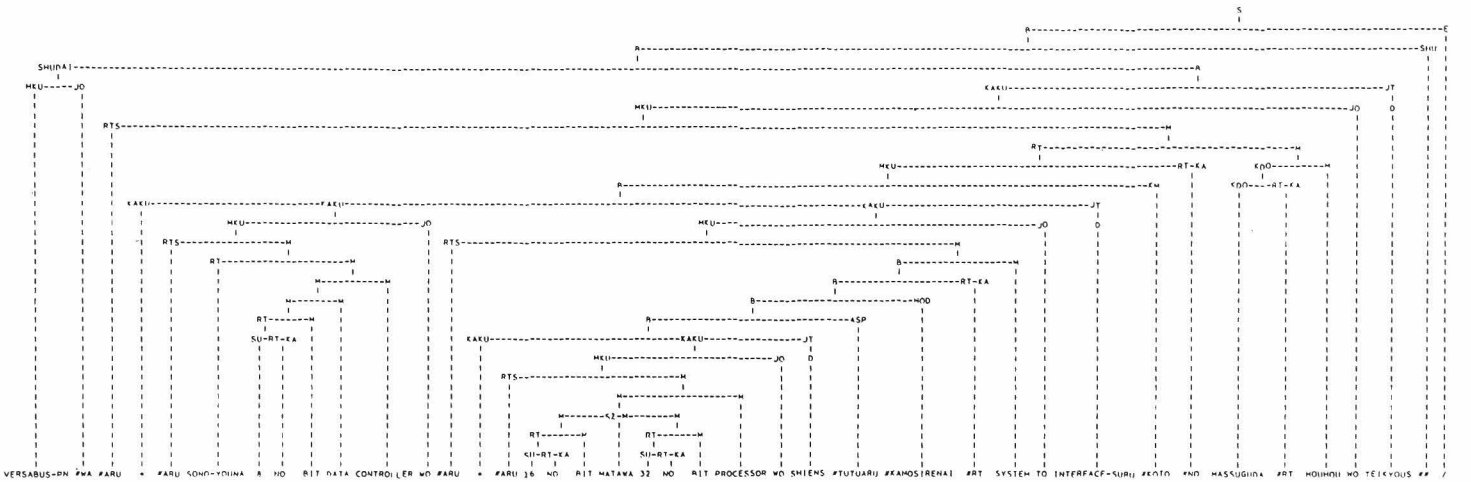
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(#DCL
  (VERSABUS-PN
    (LAMBDA
      X46
      ((A
        (LAMBDA
          X45
          ((NOM
            (LAMBDA
              X21
              ((A
                (SUCH
                  ((*ADJ-CLSF ((*ADJ-CLSF ((*NBR (QUOTE 8)) BIT)) DATA))
                    CONTROLLER)))
                (LAMBDA
                  X22
                  ((A
                    ((WHICH
                      (MAY
                        (#PROG
                          (LAMBDA
                            X9
                            ((A
                              ((*ADJ-CLSF
                                 (*OR-NOUN-NOUN
                                   ((*NBR (QUOTE 16)) BIT)
                                   ((*NBP (QUOTE 32)) BIT)))
                                PROCESSOR))
                              (LAMBDA X10 (SUPPORT-V1 X9 X10)))))))
                      SYSTEM))
                    (LAMBDA X23 ((INTERFACE-TO X21 X22 X23)))))))
            (LAMBDA X44 ((((*AP OF) X44) (STRAIGHTFORWARD WAY)) X46)))))
        (LAMBDA X47 (PROVIDE X46 X47))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



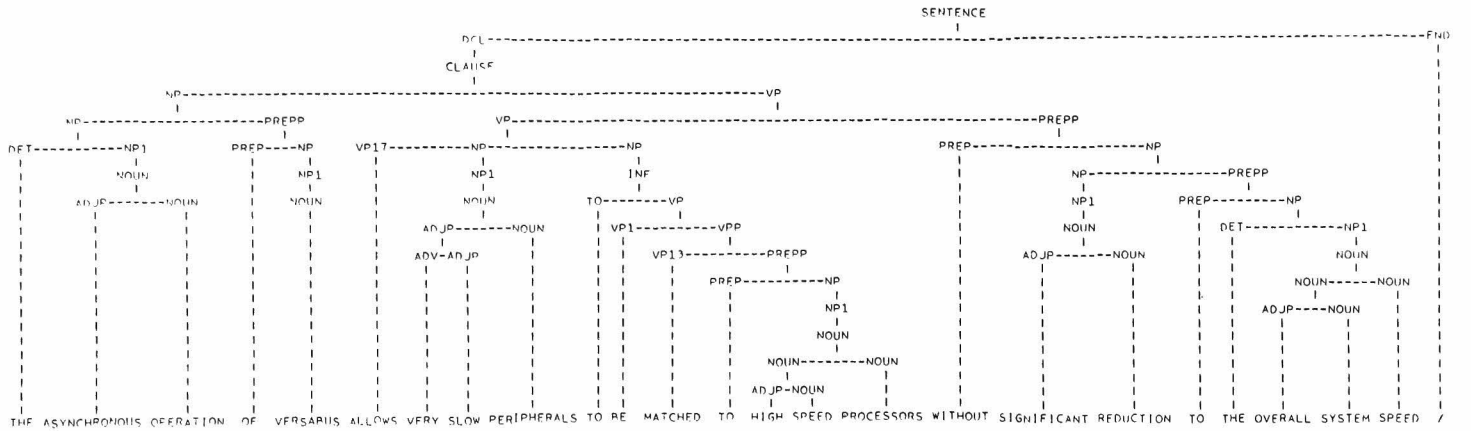
Input Sentence:

VERSABUS PROVIDES A STRAIGHTFORWARD WAY OF INTERFACING SUCH AN 8 BIT DATA CONTROLLER TO A SYSTEM THAT MAY BE SUPPORTING A 16 BIT OR 32 BIT PROCESSOR /

Output Sentence:

ＶＥＲＳＡｂｕｓはそのような８ｂｉｔデータコントローラを１６ｂｉｔまたは３２ｂｉｔプロセッサを支援しつつあるかもしれないシステムとインターフェイスで接続することのまっすぐな方法を提供する。

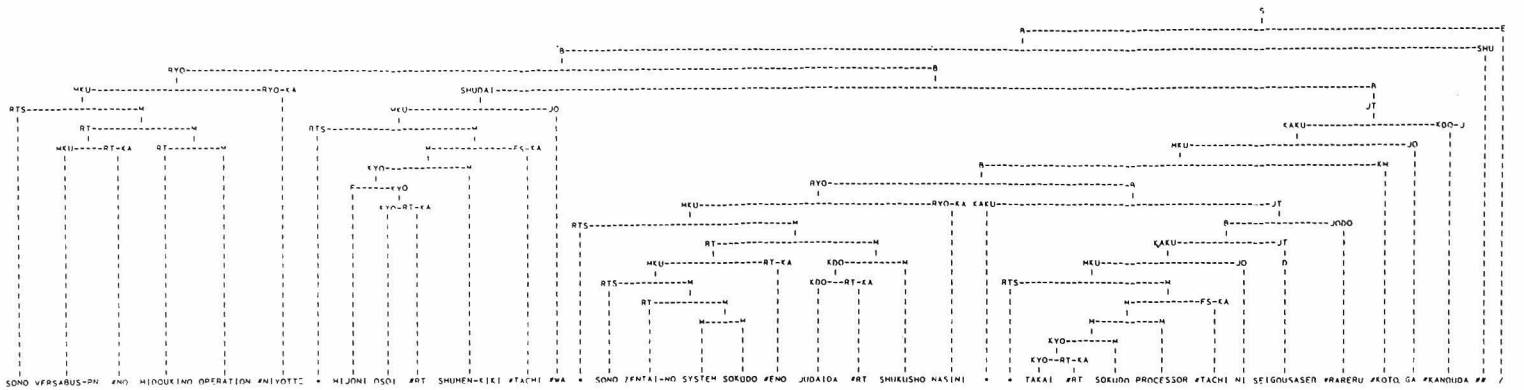
Result of Phrase Structure Analysis:



Extracted EFR Expression:

```
(*DCL
  ((THE
    (LAMBDA
      X3
      ((*AB VERSABUS)
        (LAMBDA X2 ((((*AP OF) X2) (ASYNCHRONOUS OPERATION)) X3)))))
    (LAMBDA
      X44
      ((*AB
        (LAMBDA
          X33
          ((THE ((*ADJ (OVERALL SYSTEM)) SPEED))
            (LAMBDA
              X32
              ((((*AP TO) X32) (SIGNIFICANT REDUCTION)) X33))))))
        (LAMBDA
          X43
          ((WITHOUT X43)
            ((A* (*PL ((HIGH SLOW) PERIPHERAL)))
              (LAMBDA
                X37
                ((INF-N
                  (LAMBDA
                    X34
                    ((A* (*PL ((*ADJ (HIGH SPEED)) PROCESSOR)))
                      (LAMBDA X35 ((*EN3 MATCH-TO) X34 X35)))))
                  (LAMBDA X38 (ALLOW-TO X44 X37 X38)))))))))))
```

Phrase Structure for Output Sentence (Before Heuristic Rewriting Rules are Applied):



Input Sentence:

THE ASYNCHRONOUS OPERATION OF VERSABUS ALLOWS VERY SLOW PERIPHERALS TO BE MATCHED TO HIGH SPEED PROCESSORS WITHOUT SIGNIFICANT REDUCTION TO THE OVERALL SYSTEM SPEED /

Output Sentence:

非常に遅い周辺機器はＶＥＲＳＡｂｕｓの非同期の操作によって全体のシステム速度への重大な縮小なしに高い速度のプロセッサに整合させられることが可能だ。